YAŞAR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MASTER THESIS

DATABASE OPTIMIZATION AND TUNING ON MS SQL SERVER

SUZAN ARICI

THESIS ADVISOR: ASSOC.PROF.DR. MURAT KOMESLİ

COMPUTER ENGINEERING

PRESENTATION DATE: 18.08.2017

BORNOVA / İZMİR
AUG 2017

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science / the Doctor of Philosophy.

Jury Members:	**Signature:**
Assoc. Prof. Dr. Murat KOMESLİ Yaşar University	[illegible]
Prof. Dr. Mehmet Cudi OKUR Yaşar University	[illegible]
Assoc. Prof. Dr. Çiğdem TARHAN Dokuz Eylül University	[illegible]

[illegible]

Prof. Dr. Cüneyt GUZELIS
Director of the Graduate School

ABSTRACT

DATABASE OPTIMIZATION AND TUNING ON MS SQL SERVER

Arıcı, Suzan
MSc, Computer Engineering
Advisor: Assoc. Prof. Dr. Murat KOMESLİ
Aug, 2017

In the study of MS SQL Server query optimization, index architecture is examined to understand query structure and execution. Query optimization is important for database optimization because when users examine the query and try to optimize or tune the query, they have to check all details of database to understand the reason decrease. So when fixing problem increase query performance also increase database performance. Some MS SQL Server features help users to find the problematic queries and give some advices to fix them. In addition, some Dynamic Management Views, Dynamic Management Functions and Statistics can help to see the problems on the queries.

Key Words: MS SQL Server, query optimization, optimization, query optimization, dynamic management views, dynamic management functions.

ÖZ

MS SQL SUNUMCU ÜZERİNDE VERİTABANI

ENİYİLEŞTİRME VE AYARLAMA

Arıcı, Suzan
Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Doç.Dr. Murat KOMESLİ
Ağustos 2017

MS SQL Server sorgu iyileştirme çalışmasında, sorguların yapısını ve nasıl çalıştığını anlayabilmek için indeks mimarisi incelenmektedir. Sorgu iyileştirme veri tabanı iyileştirme içinde oldukça önemlidir. Kullanıcılar sorguları iyileştirme için hata araması yaparken sorgunun veri tabanı içerisinde ulaştığı tüm detayları incelerler. Bu yüzden sorguların yavaşlama sebebi tespit edildikten sonra sorgunun iyileşmesini sağlamak aynı zamanda veri tabanını da iyileştirir. Bazı MS SQL Server'ın kendine ait özelliklerini kullanmakta sorunlu olan sorguları bulmaya yardımcı olur. Aynı zamanda bu kullanılan özellikler sorgular için yararlı tavsiyeler verebilirler. Ayrıca, dinamik yönetim görünümleri, dinamik yönetim fonksiyonları ve istatistikler de sorunlu sorguları inceleme ve iyileştirme konusunda yardımcıdırlar.

Anahtar Kelimeler: MS SQL Server, sorgu eniyileştirme, dinamik yönetim görümleri, dinamik yönetim fonksiyonları.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Assoc. Prof. Dr. Murat KOMESLİ for his guidance and patience during this study.

I would like to express my enduring love to my parents, who are always supportive, loving and caring to me in every possible way in my life.

Suzan ARICI

İzmir, 2017

[illegible]

TEXT OF OATH

I declare and honestly confirm that my study, titled "DATABASE OPTIMIZATION AND TUNING ON MS SQL SERVER" and presented as a Master's Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Suzan ARICI

[illegible]..............................

September 12, 2017

TABLE OF CONTENTS

ABSTRACT iii

ÖZ iv

ACKNOWLEDGEMENTS v

TEXT OF OATH viii

TABLE OF CONTENTS vii

LIST OF FIGURES ix

SYMBOLS AND ABBREVIATIONS xiii

CHAPTER ONE INTRODUCTION 1

1.1 INDEXES 2

1.1.1CLUSTERED INDEX 5

1.1.2NON-CLUSTERED INDEX 6

CHAPTER TWO MS SQL SERVER FEATURES 8

2.1 SQL PROFILER 9

2.2 DATABASE ENGINE TUNING ADVISOR 13

2.3 ACTIVITY MONITOR 18

2.4 EXECUTION PLAN 22

2.4.1TEXT PLAN: 22

2.4.2XML PLAN 23

2.4.3GRAPHICAL PLAN 24

CHAPTER THREE EXAMINATON OF QUERIES WITH RELATIONAL ALGEBRA .. 33

3.1 RELATIONAL ALGEBRA MAIN OPERATORS 36

3.2 QUERIES' RELATIONAL ALGEBRA OPERATIONS AND QUERY TREE REPRESENTATIONS 44

CHAPTER FOUR USING DYNAMIC MANAGEMENT VIEW (DMV) AND DYNAMIC MANAGEMENT FUNCTIONS (DMF) FOR OPTIMIZATION 47

4.1 UNUSED INDEX 48

4.2 MISSING INDEX 49

4.3 INDEX FRAGMENTATION 52

4.4 EXECUTION PLAN ANALYSIS ON CACHE 54

4.5 FINDING TOP N MOST EXPENSIVE QUERIES.. 55
4.6 THE MOST CPU CONSUMED QUERIES 56
4.7 DISK I/O BOTTLENECK 56
4.8 THE MOST EXECUTED QUERIES 57
4.9 QUERY OPTIMIZER.. 58

CHAPTER FIVE IMPORTANCE OF STATISTICS..59
5.1 DISK I/O STATISTICS 60
5.2 EXECUTION TIME STATISTICS 61
5.3 MONITORING STATISTICS 63

CHAPTER SIX RELATED WORKS...66

CHAPTER SEVEN CONCLUSIONS AND RESULTS 71

REFERENCES...73

APPENDIX 1 – DENTAL CLINIC DATABASE SQL CODE 76

LIST OF FIGURES

Figure 1. SQL Server Balanced Tree Index Structure (www.red-gate.com/simple-talk/sql/database-administration/brads-sure-guide-to-indexes/).............................. 3

Figure 2. B-Tree Structure (Adar, 2016).. 4

Figure 3. With using Clustered Index, Leaf node has own data row (www.red-gate.com/simple-talk/sql/database-administration/brads-sure-guide-to-indexes/)... 5

Figure 4. Non-Clustered index has pointers for locate the data (www.red-gate.com/simple-talk/sql/database-administration/brads-sure-guide-to-indexes/).............................. 7

Figure 5. SQL Profiler place. 10

Figure 6. Trace file options. 10

Figure 7. Necessary performance selection for recorded on trace file................................ 11

Figure 8. Appearance of the Trace file.. 12

Figure 9. Appearance of Database Engine Tuning Advisor.. 13

Figure 10. Tuning Options' Selections. 14

Figure 11. Recommendations Page of Database Engine Tuning Advisor. 14

Figure 12. Analyzing of Third Query. 15

Figure 13. The Performance Impact of Index. 16

Figure 14. Index recommendation for Query Four. 17

Figure 15. The Performance Impact of Index. 17

Figure 16. Display of Overview Pane (mssqldude.wordpress.com, 2011)........................... 18

Figure 17. Display of Process Pane. 19

Figure 18. Selections of Processes Pane. 19

Figure 19. Display of Resources Waits Pane. 19

Figure 20. Display of Data File I/O. 20

Figure 21. Display of Recent Expensive Queries Pane... 21

Figure 22. Right Click Options of Expensive Query. 21

Figure 23. Edit Query Selection... 21

Figure 24. Execution Plan Selection. 22

Figure 25. Display of Text Plan. 23

Figure 26. XML Plan File. 23

Figure 27. Display XML Plan as a Graph. 24

Figure 28. Execute Query with Execution Plan Selection. 24

Figure 29. Execution Plan of Fourth Query. 26

Figure 30. Display of Icons' Tooltips. 26

Figure 31. Recommended Indexes. 27

Figure 32. Non-clustered index applied Person.Person table. 28

Figure 33. Tooltips After First Index on Person.person table. 28

Figure 34. Non-clustered index applied Sales.Customer table. 29

Figure 35. Tooltips after Second Applied Index on Sales.Customer table. 29

Figure 36. Execution Plan of Fifth Query. 30

Figure 37. Tooltips of Operators of Query five. 30

Figure 38. Recommendations of Data Base Engine Tuning Advisor. 31

Figure 39. All Recommendations are Applied on The Query. 31

Figure 40. Tooltips of Operators. 32

Figure 41. Place of Relational Algebra Query in the Query Optimization (Ricardo, Urban, 2016). 34

Figure 42. Class Diagram of Dental Clinic Database. 35

Figure 43. ER Diagram of Dental Clinic Database. 35

Figure 44. Dental Clinic Database Diagram from MS SQL Server. 36

Figure 45. Representation of Query Result. 38

Figure 46. Representation of Query Result. 39

Figure 47. Representation of Query Result. 40

Figure 48. Representation of Query Result. 41

Figure 49. Query Tree of name,surname(σage=18 and gender=F(Patient ⋈ Dentist)) 42

Figure 50. Query Tree of Πp.name,p.age,d.name,s.description(σ (Patient⨝Dentist⨝ Specialty)(Πsurname(σgroup by surname>1(Patient)))) 42

Figure 51. Query Tree of Πp.Name,p.Surname,p.Age,p.Gender,a.Description(σ (Patient⨝ Appointment)(ΠDentistID(σSpecID=1(Dentist)))) 43

Figure 52. Query Tree of Πname,age,gender(σDentistID=654321(Patient))...................... 43

Figure 53. Query Tree of Query Three. 45

Figure 54. Query Tree of Query Four. 45

Figure 55. Query Tree of Query Five. 46

Figure 56. Querying of Unused Index. 48

Figure 57. Querying of Unused index.. 49

Figure 58. Missing Index Warnings on Query Execution Plan Graph................................ 50

Figure 59. Missing Index Querying. 51

Figure 60. DMV Missing Index Querying. 51

Figure 61. Finding Fragmented Index.. 53

Figure 62. Fixing Fragmentation. 53

Figure 63. Execution Plan Analysis on Cached. 54

Figure 64. Most Expensive Queries. 55

Figure 65. CPU Consumed Queries. 56

Figure 66. The most I/O Perform Queries. 57

Figure 67. The Most Executed Queries List. 58

Figure 68. Optimizer Information. 58

Figure 69. Statistics of Person.Person Table... 59

Figure 70. Query Four and Query Five I/O Statistics before Index. 60

Figure 71. I/O Statistics after Index. 61

Figure 72. Execution Time Statistics before Index. 62

Figure 73. Execution Time Statistics after Index Applied. 63

Figure 74. Query Four and Five Without Index. 64

Figure 75. Query Four and Five With Index. 65

Figure 76. Display of RelaX. 66

Figure 77. Display of RAT. 67

Figure 78. Display of MS SQL Sensor (www.paessler.com/database-monitoring). 67

Figure 79. Display of SQL Assistant on Databases (www.softtreetech.com/sqlassist/index.htm). 68

Figure 80. Display of SolarWinds (www.solarwinds.com/database-performance-analyzer). 69

Figure 81. Display of Free Sql Performance Monitoring Tool (www.manageengine.com/sql-performance-monitor). 70

SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS:

MS SQL	Microsoft Structured Query Language
DMV	Dynamic Management Views
DMF	Dynamic Management Functions

CHAPTER ONE
INTRODUCTION

This study aims that to explain what query optimization is in MS SQL Server with understandable way and it will provide information on how to use certain features of the MS SQL Server to improve performance, how to interpret the results, and how to improve the query.

First, it needs to know why query optimization is necessary. With right optimization, query result can be reached quickly, but primarily, queries should be monitored and examined to understand their needs.

Getting fast results from the query is the goal of the query optimizing. In the MS SQL server, users can use different methods for getting better results. Using indexes can decrease search time to bring queries' information, and monitor queries with activity monitor; it gives recent expensive queries' informations, disk I/O and CPU usage so user can interfere these queries. MS SQL profiler can create trace file for query and then, this file is used by database engine tuning to analysis and then it gives some recommendations if it is necessary for a given query. In addition, there is an execution plan option on SQL server, it shows a query execution plan and gives information about the query.

This work will be done with indexes, SQL Server own features, Dynamic Management Views and Dynamic Management Functions and statistics. Firstly, indexes and their architecture will be presented. So, users are expected to know what is an index? How can SQL Server use them? Moreover, how they work in the query. Then, SQL Server Features will be introduced and their missions, and Dynamic Management Views, which are the same functions with these features, but they have only one difference; they have own queries. This study has seven chapters;

Aim of the study, which MS SQL Server features will be used on this study, and importance of the index is explained in chapter one. Chapter two examines some importance MS SQL Server features that are MS SQL Profiler, Database Engine

Tuning Advisor, Activity Monitor and Execution Plan and their usage, using AdventureWorks2014 database's some queries are created to do performance analysis and query optimization. To understand query execution on query optimizer, relational algebra job need to be known, chapter three explains the relational algebra function and query tree. Dynamic Management Views and Dynamic Management Functions are important for monitoring performance so *chapter four* mentioned these two for optimization. Statistics can be used query and performance optimization some statistical information give important clues to users, *chapter five* explains these statistics. Definitely, there are different tools that can be used with MS SQL Server to monitor and optimize query performance, *chapter six* includes these related works. All of these optimization options are used on queries that have poor performance, and some results have been reached these optimization jobs and study result mentioned on in *chapter seven*.

1.1 INDEXES

Indexes are used to read queries quickly in crowded databases. When running queries needed to pull data from the table are used to reduce the time required. Maybe indexes are not necessary for small databases but when working with big databases, indexes can be lifesaving (Gözüdeli, 2014). By all means, using index cannot always increase the performance, some usage of indexes can be wrong or unnecessary, so it reduces query performance.

First, users have to have an idea about the way of work of index. It can be explained through an example; assume there is a bookstore and a customer wants 'The Phantom of the Opera' bookseller starts scanning all stack without any order, this is table scan, it is not good for time and customer. If bookseller orders books with their names, when a customer asks 'The Phantom of the Opera', he/she directly goes that bookshelf and finds the book. So he/she and customer save time, it is a clustered index. However, if a customer only asks the author name 'Gaston Leroux- The Phantom of the Opera' (assume there are some several books of the same name), booksellers system will not work well against to this want. Nevertheless, if bookseller orders books with author names, he/she will go the bookshelf that starts with 'G' and then he/she will find easily, it is non-clustered index. Before index type and index architecture, users need to know some terminology of index:

Seek Operation: SQL Server reaches information directly with using B-Tree structure. It is more preferable, and faster than scan operation.

Scan Operation: It scans all data set to get information. Table scan operation is the slowest one.

Lookup Operation: When cannot be reached information by using index, lookup operation reach the information with the help of index and identified key value.

Actually, clearly understand to how the index works B-tree structure need to be learnt. Therefore, at the beginning, index architecture will be trying to understand.

Index Architecture: Balanced Tree Structure (B-Tree Structure):

To understand index working system Balanced Tree structure will be examined, In SQL Server indexes are organized into a B-tree structure. Balanced tree comprises of root level, intermediate level and leaf level. Figure 1 shows SQL Server balanced tree structure.

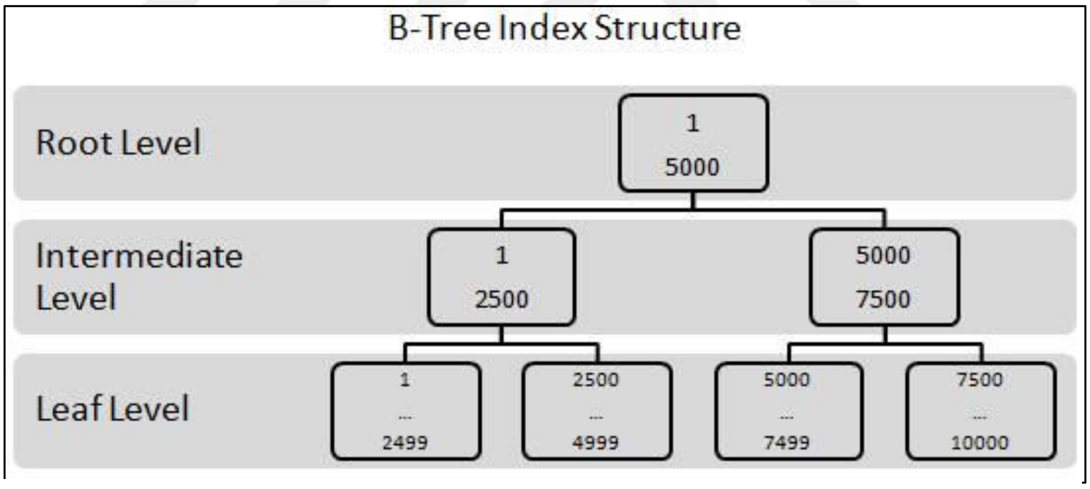


Figure 1. SQL Server Balanced Tree Index Structure (www.red-gate.com/simple-talk/sql/database-administration/brads-sure-guide-to-indexes/).

<u>Root Level (Level 2):</u> This level (node) keeps all information about our data. SQL Server starts searching information from here. Also branching starts here by nature of balanced tree structure.

<u>Intermediate Level (Level 1) (Non-Leaf):</u> Searching continues with intermediate level this level may have more than one level depends on a number of records and index size (Adar, 2016). Therefore, can be said that a root level has one or more intermediate level.

<u>**Leaf Level (Level 0):**</u> Data searching ends this level. It starts with root level and continues with intermediate levels after eliminating it reaches the leaf level it keeps data in different way depends on index type.

For instance; in the bookseller example, the customer wants to buy ‘Phantom of Opera’, how bookseller finds this book with a B-Tree structure will be examined with helping of Figure 2.

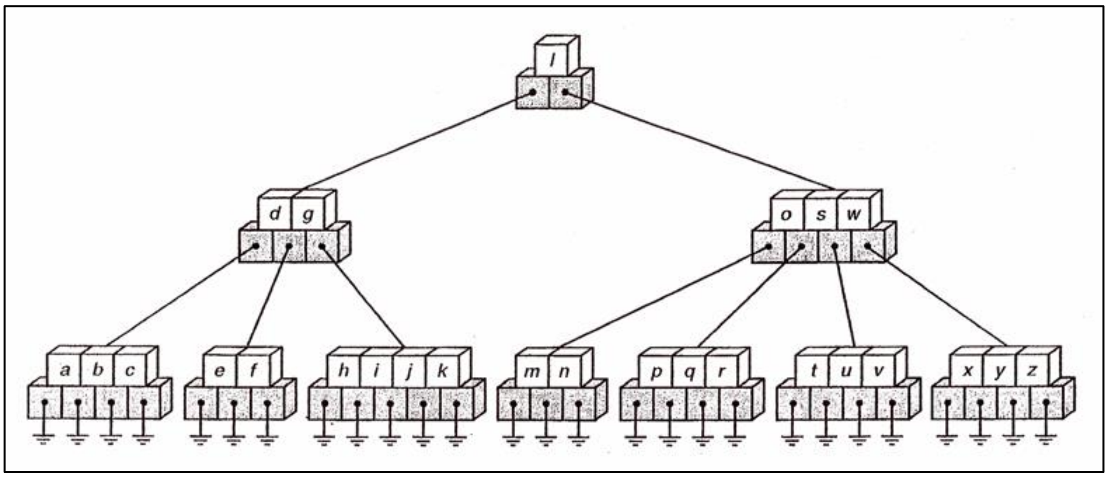


Figure 2. B-Tree Structure (Adar, 2016).

Data information searching starts with root level users compare their information with this root’s information. Is P smaller than L, if it is users will continue their searching with left intermediate leaf, but it is greater than L, they keep their searching with right intermediate leaf. It is greater than L, searching moves on the right side. Therefore, again, they compare P and O, S and W, and users see P is greater than O so they will look right side and P is smaller than S they look left they get P, Q, R leaf nodes it keeps related data. Their searching ends, leaf level for B-Tree. Finding data changes according to index types.

Without index, SQL Server search all records without any order and checks all data even if it finds the data against the situation of having same data it is called **Table Scan (Full Table Scan)**. Also, It must be known that data records (if not specified) randomly settled on the table, this structure is called **Heap Structure**. In parallel with this structure, table scan may be expensive (time-consuming) for finding a specific data in the thousands and millions data. In addition to this situation, indexes are used for find a data easily and save the time.

1.1.1 Clustered Index

Clustered index is also known as physical index because of data that are physically sorted according to index defined column. For this reason, only one clustered index can be defined for a table. In Figure 2 shows how B-tree structure works on SQL Server, to find data from the example, reaching data at the leaf node that point, if this node has all information about searching data it is called clustered index. Figure 3 shows clustered index b-tree structure.

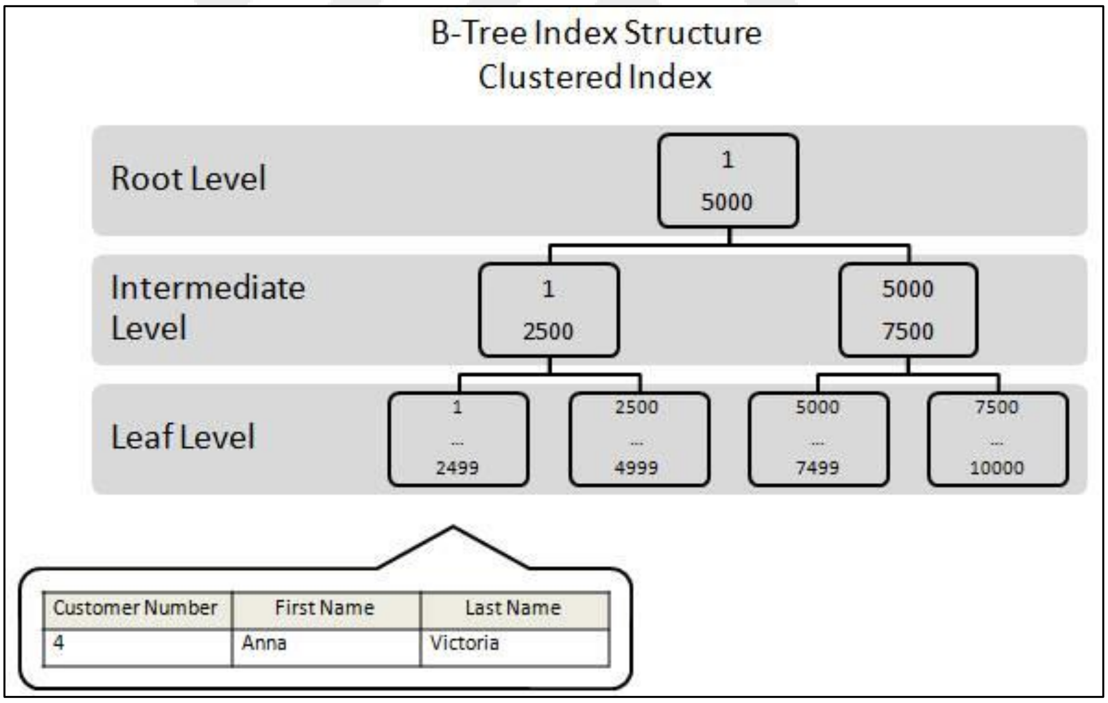


Figure 3. With using Clustered Index, Leaf node has own data row (www.red-gate.com/simple-talk/sql/database-administration/brads-sure-guide-to-indexes/).

Any changed on the one of clustered index column can cause physically reorganizing data on the table. That cause loss of performance for big tables (Adar, 2016).

If there are any restriction on the query, clustered index used as clustered index seek. If there are not restriction on the query, clustered index used as clustered index scan.

Creating clustered index with T-SQL:

```
CREATE CLUSTERED INDEX[Index_Name] ON
[DataBase_Name].[Schema_Name]
(
        [Column_Name] ASC/DESC
)
```

1.1.2 Non-Clustered Index

Non-Clustered indexes are ordered by logically, it means that; opposite of the clustered index, non-clustered index does not have all information about the data on the leaf node; it has an information about where data is located. Therefore, non-clustered index cannot reach data directly. Non-clustered index reaches data over the clustered index and heap.

If non-clustered index defined on a table that has clustered index, this type of index has key value. Key value is a reference for clustered index (Gözüdeli, 2014). These are shown on Figure 4.

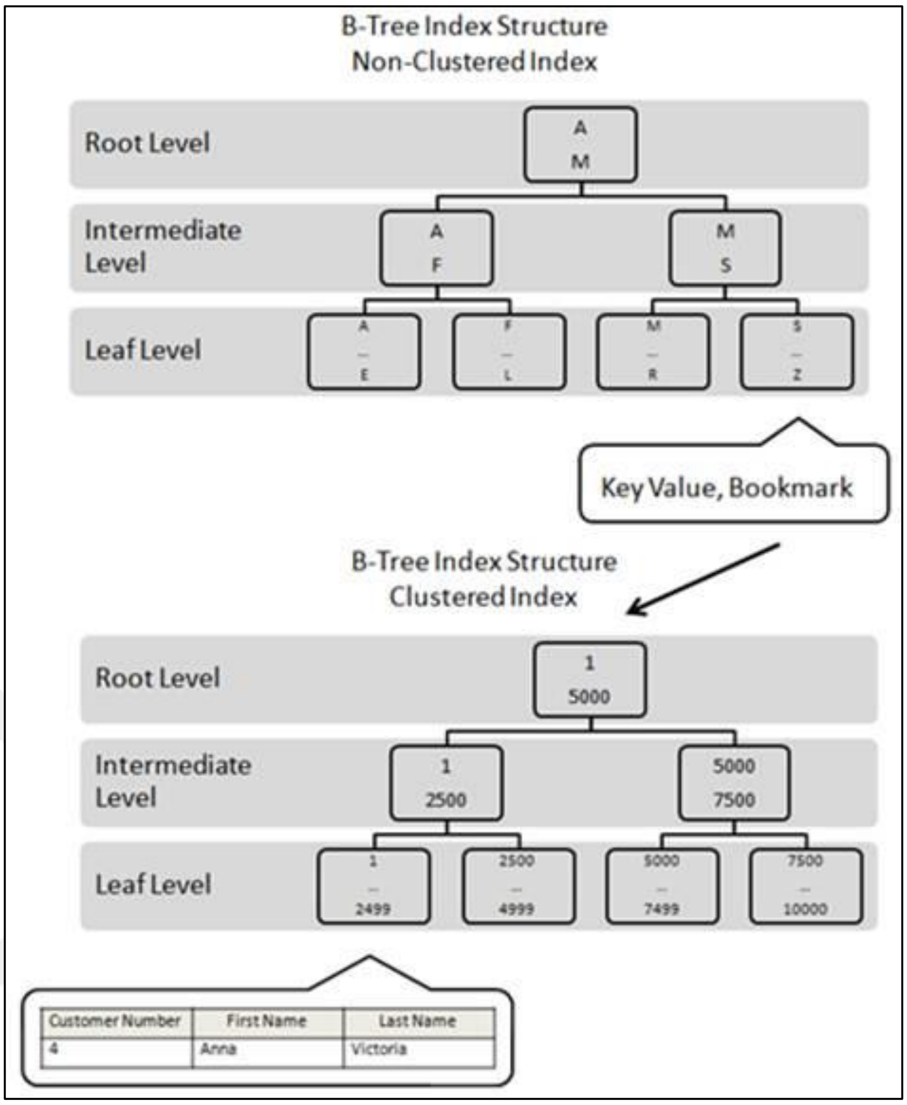


Figure 4. Non-Clustered index has pointers for locate the data (www.red-gate.com/simple-talk/sql/database-administration/brads-sure-guide-to-indexes/).

Non-clustered index is fast about reaching data but it can be used with clustered index seek or clustered index scan so its performance is lower than clustered index (Gözüdeli, 2014).

Creating non-clustered index with T-SQL:

```
CREATE NONCLUSTERED INDEX[Index_Name] ON
[DataBase_Name].[Schema_Name]
(
[Column_Name] ASC/DESC
)
```

CHAPTER TWO
MS SQL SERVER FEATURES

MS SQL Server has some own features; users can control and check their SQL query performances. In this study, SQL Profiler, Database Engine Tuning Advisor, Activity Monitor and Execution Plan will be examined. With all of these features, queries can be followed and user can find necessity of query.

First, it needs to known that AdventureWorks2014 Database of MS SQL Server will be used and following queries related with this database.

Queries and Meanings

Queries are:

1. select *

   from person.Person

This query result shows all information on Person.Person table.

2. select *

   from sales.Customer

This query result shows all information on Sales.Customer table.

3. Select StoreID

   From Sales.Customer

   Where StoreID Is Not Null and AccountNumber like '%17'

This query result shows StoreID which is not null and include 17 in AccountNumber from Sales.Customer with condition of Where

4. Select PersonID,Title,FirstName,LastName

   From Sales.Customer As s Inner Join

   Person.Person As p On p.BusinessEntityID = s.PersonID

   Where PersonID Is Not Null

This query will bring PersonId column from Sales.Customer table and Title, FirstName and LastName of customers from Person.Person table so it is used Inner Join to associate two different table and there is a Where condition and it does not want empty PersonId.

5. Select StoreID, FirstName,MiddleName,LastName
   From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
   Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000

This query result shows StoreID from Sales.Customer table, FirstName, MiddleName, LastName from Person.Person table so it is used Join to associate two different table and there is a Where condition and it does not want empty StoreID and will be smaller than 1000, in addition AccountNumber will include 17.

2.1 SQL PROFILER

SQL Profiler is the one of the SQL Server's tools. Profiler monitor events on the SQL Server. It helps to performance monitoring. Users can analyze the SQL Server performance and statistics (Gözüdeli, 2014). With SQL Profiler, they can analyze query plans of most useful queries, and find solution for performance optimization (Sahtiyan, 2011b).

Showing Usage of SQL Profiler with Pictures

First, SQL Server Profiler should be selected from Tools selection (Figure 5):

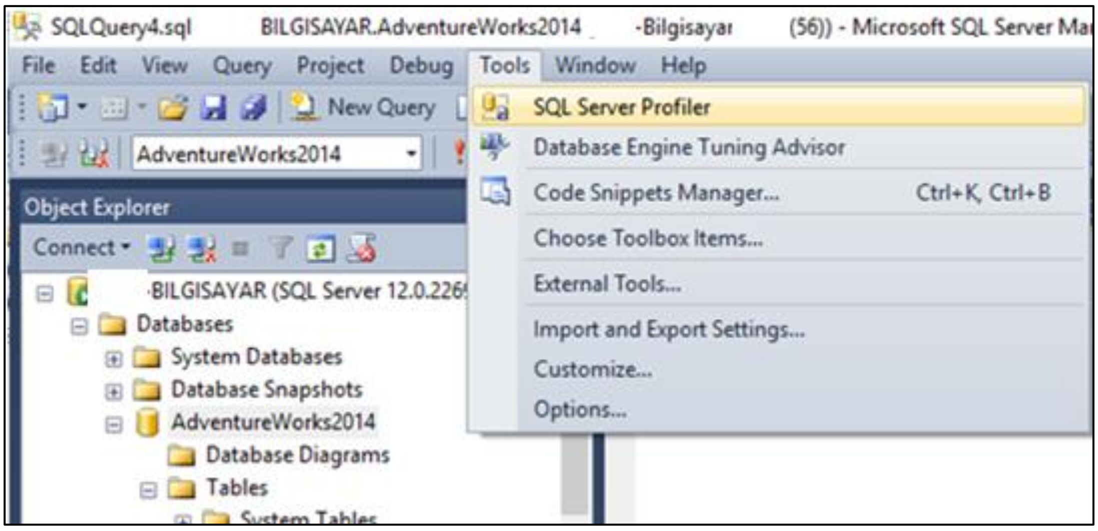
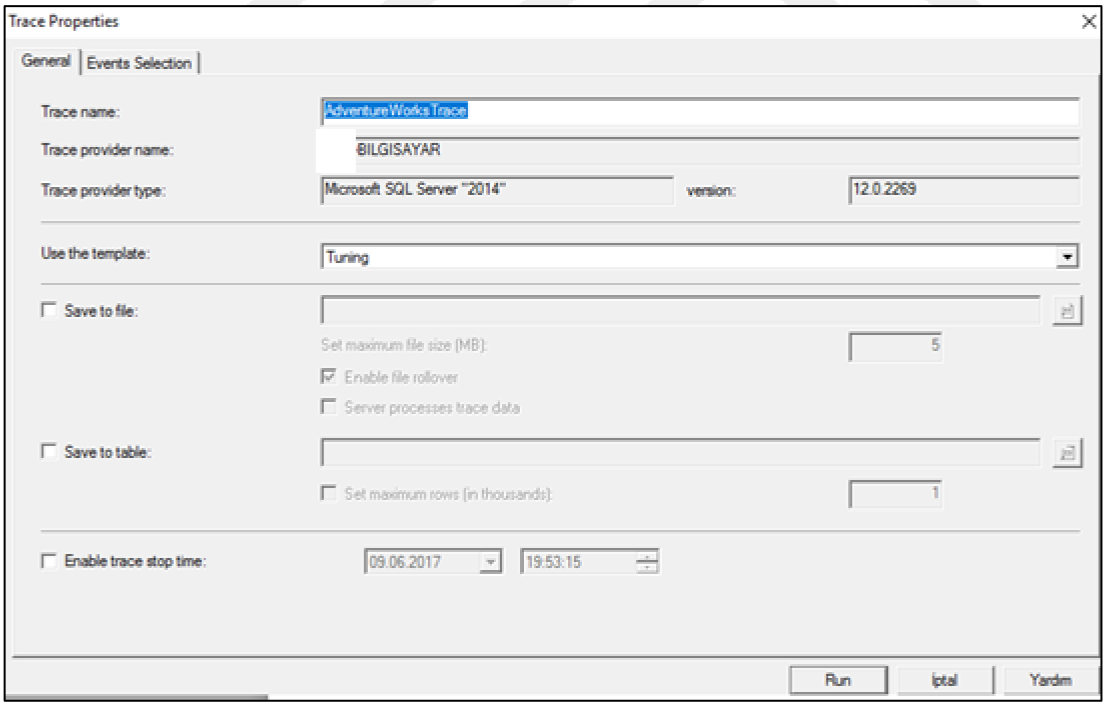

SQLQuery4.sql BILGISAYAR.AdventureWorks2014 -Bilgisayar (56)) - Microsoft SQL Server Ma
File Edit View Query Project Debug Tools Window Help
New Query
AdventureWorks2014
SQL Server Profiler
Database Engine Tuning Advisor
Code Snippets Manager... Ctrl+K, Ctrl+B
Choose Toolbox Items...
External Tools...
Import and Export Settings...
Customize...
Options...
Object Explorer
Connect
BILGISAYAR (SQL Server 12.0.226
Databases
System Databases
Database Snapshots
AdventureWorks2014
Database Diagrams
Tables
System Tables

Figure 5. SQL Profiler place.

Then, user has to choose server after choosing server should be given a name for trace file (Figure 6):

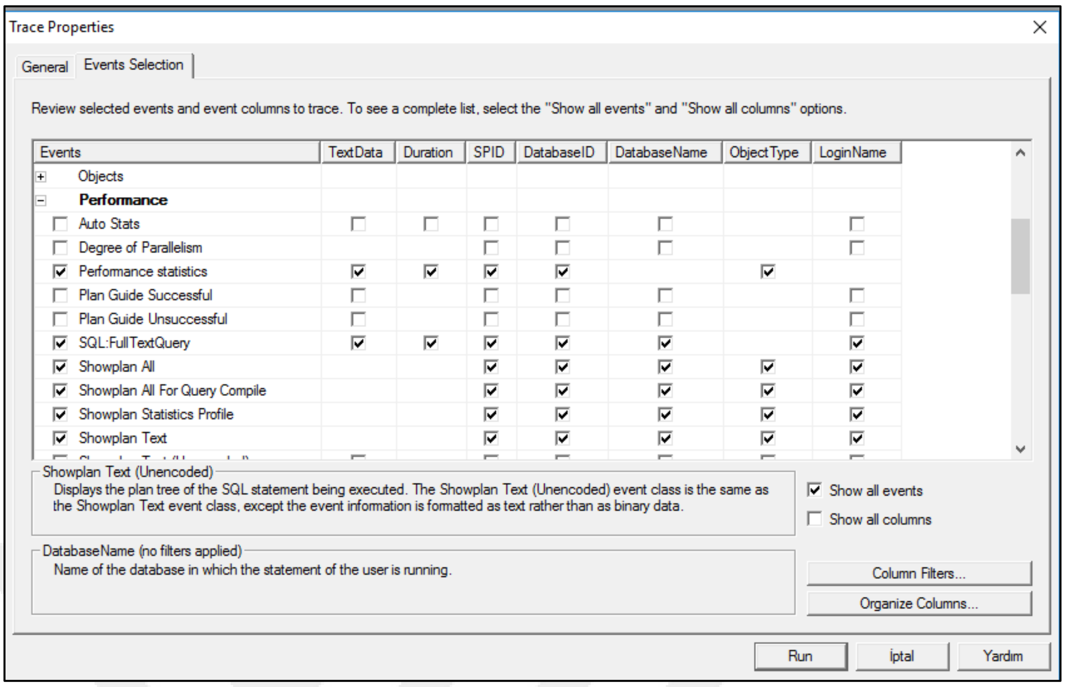

Trace Properties

General | Events Selection |

Trace name:	AdventureWorksTrace		
Trace provider name:	BILGISAYAR		
Trace provider type:	Microsoft SQL Server "2014"	version:	12.0.2269
Use the template:	Tuning		
☐ Save to file:			
	Set maximum file size (MB):		5
	☑ Enable file rollover		
	☐ Server processes trace data		
☐ Save to table:			
	☐ Set maximum rows (in thousands):		1
☐ Enable trace stop time:	09.06.2017	19:53:15	

Figure 6. Trace file options.

It can be selected necessary units from Event Selection part, then click run button (Figure 7):



Review selected events and event columns to trace. To see a complete list, select the "Show all events" and "Show all columns" options.

Events	TextData	Duration	SPID	DatabaseID	DatabaseName	ObjectType	LoginName
+ Objects							
− **Performance**							
☐ Auto Stats	☐	☐	☐	☐	☐		☐
☐ Degree of Parallelism			☐	☐	☐		☐
☑ Performance statistics	☑	☑	☑	☑		☑	
☐ Plan Guide Successful	☐		☐	☐	☐		☐
☐ Plan Guide Unsuccessful	☐		☐	☐	☐		☐
☑ SQL:FullTextQuery	☑	☑	☑	☑	☑		☑
☑ Showplan All			☑	☑	☑	☑	☑
☑ Showplan All For Query Compile			☑	☑	☑	☑	☑
☑ Showplan Statistics Profile			☑	☑	☑	☑	☑
☑ Showplan Text			☑	☑	☑	☑	☑

Showplan Text (Unencoded)
Displays the plan tree of the SQL statement being executed. The Showplan Text (Unencoded) event class is the same as the Showplan Text event class, except the event information is formatted as text rather than as binary data.

[✓] Show all events
[ ] Show all columns

DatabaseName (no filters applied)
Name of the database in which the statement of the user is running.

Figure 7. Necessary performance selection for recorded on trace file

After clicking run button trace file begins to record data, so to get good results trace file should run and record data at least one day. Some queries will be used for trace file:

1. Select *

   From person.Person

2. Select *

   From sales.Customer

3. Select PersonID,Title,FirstName,LastName

   From Sales.Customer As sc

   Inner Join

   Person.Person As pp On pp.BusinessEntityID = sc.PersonID

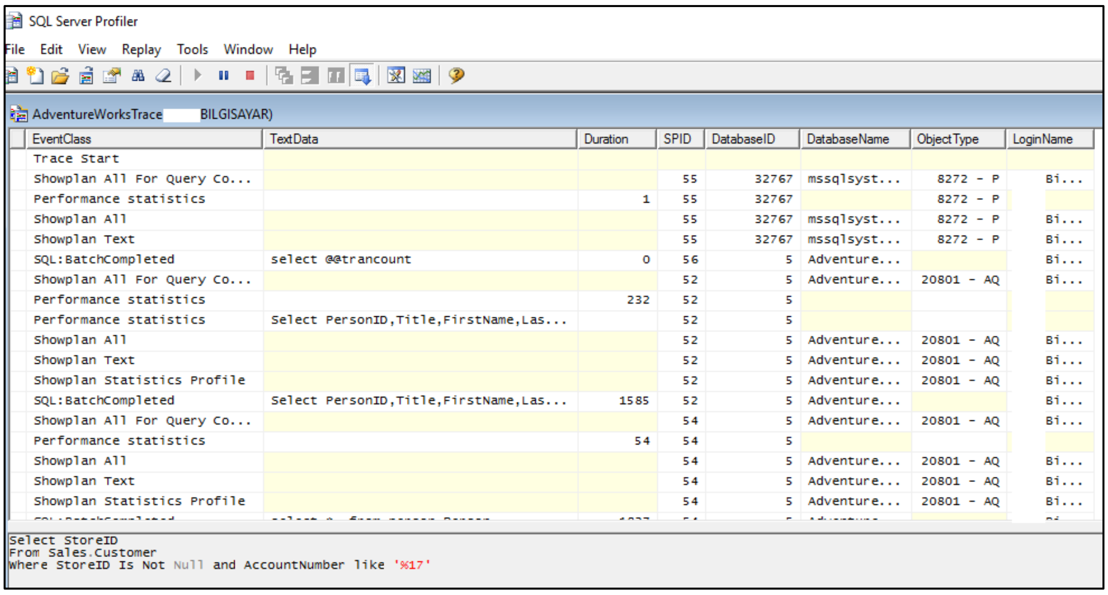
   Where PersonID Is Not Null

4. Select StoreID

   From Sales.Customer

   Where StoreID Is Not Null and AccountNumber like '%17'

In addition, view of trace file (Figure 8):


SQL Server Profiler
File Edit View Replay Tools Window Help
AdventureWorksTrace [redacted] BILGISAYAR)

EventClass	TextData	Duration	SPID	DatabaseID	DatabaseName	ObjectType	LoginName
Trace Start							
Showplan All For Query Co...			55	32767	mssqlsyst...	8272 - P	Bi...
Performance statistics		1	55	32767		8272 - P	
Showplan All			55	32767	mssqlsyst...	8272 - P	Bi...
Showplan Text			55	32767	mssqlsyst...	8272 - P	Bi...
SQL:BatchCompleted	select @@trancount	0	56	5	Adventure...		Bi...
Showplan All For Query Co...			52	5	Adventure...	20801 - AQ	Bi...
Performance statistics		232	52	5			
Performance statistics	Select PersonID,Title,FirstName,Las...		52	5			
Showplan All			52	5	Adventure...	20801 - AQ	Bi...
Showplan Text			52	5	Adventure...	20801 - AQ	Bi...
Showplan Statistics Profile			52	5	Adventure...	20801 - AQ	Bi...
SQL:BatchCompleted	Select PersonID,Title,FirstName,Las...	1585	52	5	Adventure...		Bi...
Showplan All For Query Co...			54	5	Adventure...	20801 - AQ	Bi...
Performance statistics		54	54	5			
Showplan All			54	5	Adventure...	20801 - AQ	Bi...
Showplan Text			54	5	Adventure...	20801 - AQ	Bi...
Showplan Statistics Profile			54	5	Adventure...	20801 - AQ	Bi...

```
Select StoreID
From Sales.Customer
Where StoreID Is Not Null and AccountNumber like '%17'
```

Figure 8. Appearance of the Trace file.

<u>EventClass</u> shows the type of query,

<u>TextData</u> shows the query,

<u>Duration</u> shows the cost of query execution,

<u>SPID</u> shows the users code (Gözüdeli, 2014).

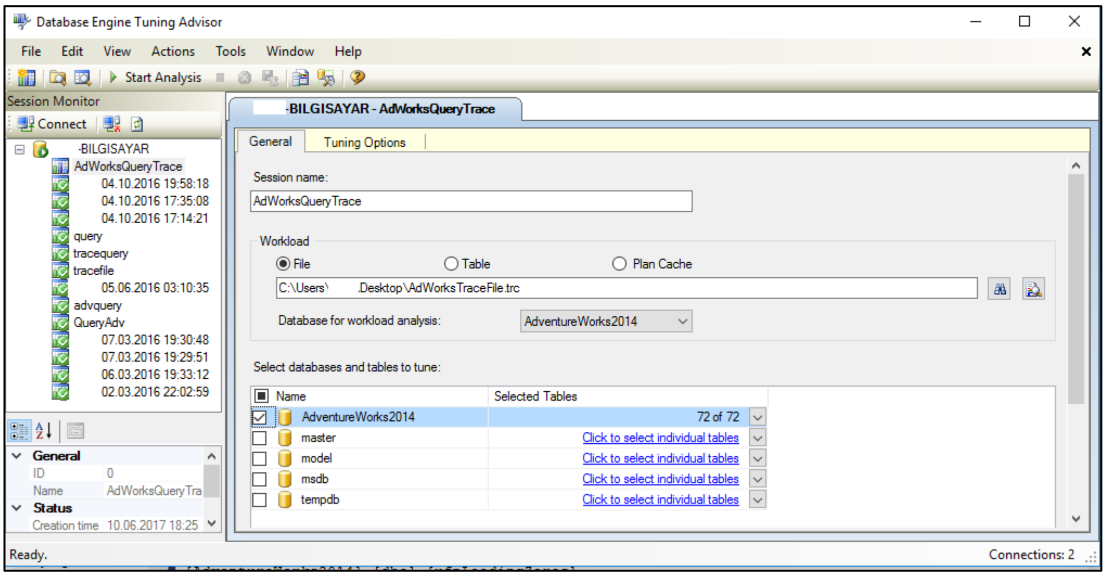
Users can find best query for same result with comparing their durations. To stop trace file, user must click red square button and save as trace file. Because Database Engine Tuning Advisor tool uses this file for analyze queries.

2.2 DATABASE ENGINE TUNING ADVISOR

The other SQL Server performance-monitoring tool is Database Engine Tuning Advisor. To use the tool users need to create trace file like created on SQL Profiler or it can save the query as a trace file. Nevertheless, using SQL Profiler trace file is most preferable because it records all events when users use database/server for long time by this way, users can get better recommendations from Database Engine Tuning Advisor. These recommendations can be; index recommendations, view recommendations or partitioning recommendations.

Database Engine Tuning Advisor placed at tool button, to use this tool (Figure 9):

- First users should give a name their session,
- Select workload type and select trace file,
- Select Database for workload,
- Choosing Database for tuning,
- Users can make the necessary adjustment from Tuning Options part (Figure 10),
- Finally, click Start Analysis button to start process.



-BILGISAYAR - AdWorksQueryTrace

-BILGISAYAR
AdWorksQueryTrace
04.10.2016 19:58:18
04.10.2016 17:35:08
04.10.2016 17:14:21
query
tracequery
tracefile
05.06.2016 03:10:35
advquery
QueryAdv
07.03.2016 19:30:48
07.03.2016 19:29:51
06.03.2016 19:33:12
02.03.2016 22:02:59

General	
ID	0
Name	AdWorksQueryTra
Status	
Creation time	10.06.2017 18:25

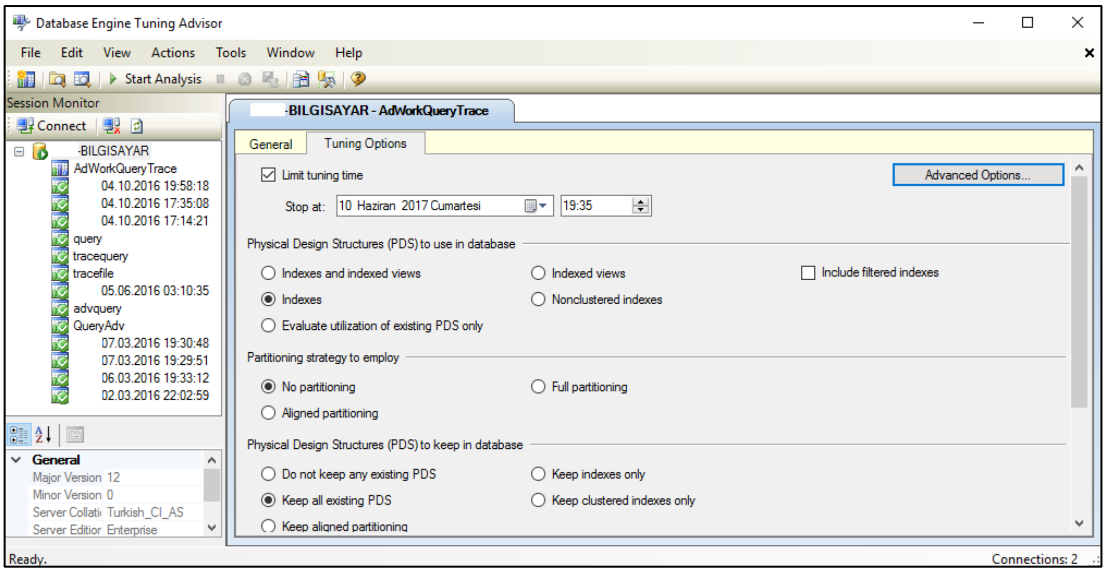
Session name: AdWorksQueryTrace

Workload
(● File) ○ Table ○ Plan Cache
C:\Users\ .Desktop\AdWorksTraceFile.trc
Database for workload analysis: AdventureWorks2014

Select databases and tables to tune:

Name	Selected Tables
✓ AdventureWorks2014	72 of 72
master	Click to select individual tables
model	Click to select individual tables
msdb	Click to select individual tables
tempdb	Click to select individual tables

Figure 9. Appearance of Database Engine Tuning Advisor.



[✓] Limit tuning time

Stop at: 10 Haziran 2017 Cumartesi 19:35

Physical Design Structures (PDS) to use in database

- ○ Indexes and indexed views
- ○ Indexed views
- ☐ Include filtered indexes
- [✓ Indexes]
- ○ Nonclustered indexes
- ○ Evaluate utilization of existing PDS only

Partitioning strategy to employ

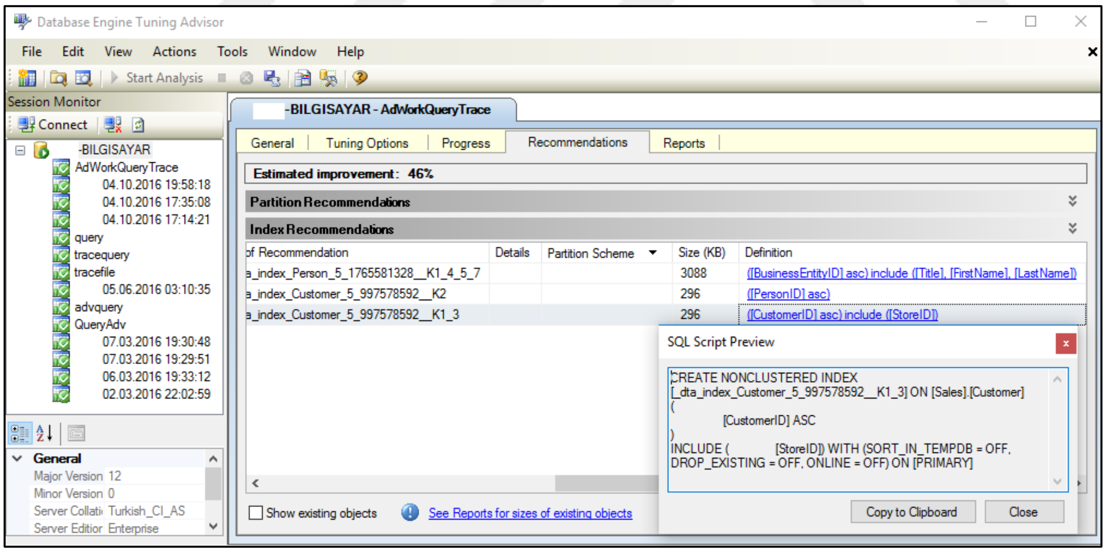
- [✓ No partitioning]
- ○ Full partitioning
- ○ Aligned partitioning

Physical Design Structures (PDS) to keep in database

- ○ Do not keep any existing PDS
- ○ Keep indexes only
- [✓ Keep all existing PDS]
- ○ Keep clustered indexes only
- ○ Keep aligned partitioning

Figure 10. Tuning Options' Selections.

After analyzing Database Engine Tuning Advisor displays recommendations page, recommended indexes and estimated improvement can be seen in this page, also users can apply directly these recommended indexes (Figure 11):



Estimated improvement: 46%

of Recommendation	Details	Partition Scheme	Size (KB)	Definition
a_index_Person_5_1765581328__K1_4_5_7			3088	([BusinessEntityID] asc) include ([Title], [FirstName], [LastName])
a_index_Customer_5_997578592__K2			296	([PersonID] asc)
a_index_Customer_5_997578592__K1_3			296	([CustomerID] asc) include ([StoreID])

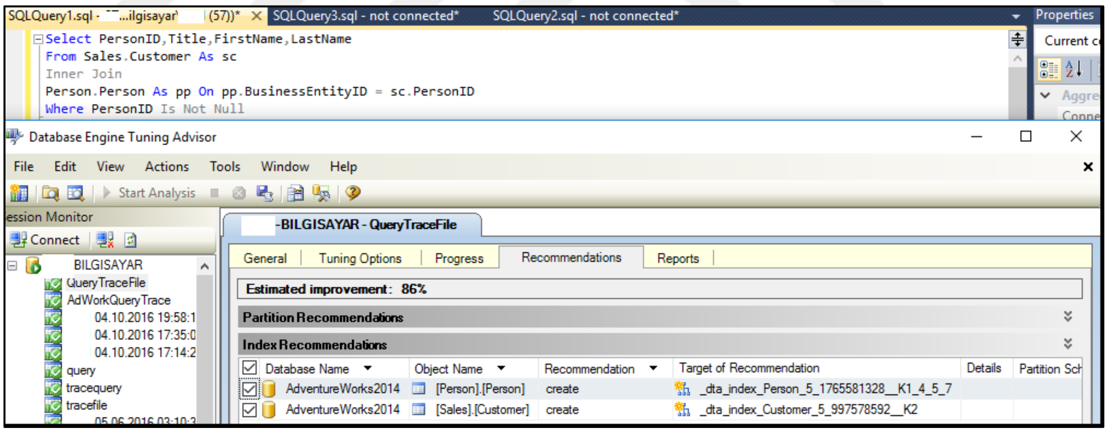
```
CREATE NONCLUSTERED INDEX
[_dta_index_Customer_5_997578592__K1_3] ON [Sales].[Customer]
(
        [CustomerID] ASC
)
INCLUDE (       [StoreID]) WITH (SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
```

Figure 11. Recommendations Page of Database Engine Tuning Advisor.

For all trace file, Database Engine Tuning Advisor gives different indexes recommendations with estimated improvement. It should not forget, it is just estimate not guarantee improvement.

Applying all recommended indexes without control is not useful. Therefore, some examples will be shown about improving performance. Created two different trace files which are recorded different queries, it will be shown and explained with pictures, recommended indexes how effect our performance.

First recommended indexes, for AdventureWorksTrace (Figure5) file which is previously created on SQL Profiler. Moreover, Tuning Advisor gives some index recommendations for this file with 46% estimated improvement. However, AdventureWorksTrace file recorded all four queries that are mentioned on Showing Usage of SQL Profiler with Pictures part. When Database Engine Tuning Advisor analyzed this trace file, it gives three different index recommendation for third and fourth queries. If this recommended indexes applied on these two queries, it may affect performance negatively because sometime using index can create undesirable results. Therefore, users have to be sure about the reliability of the result. For this reason, two different trace files created that are third and fourth queries separated and then these trace files analyzed with the Database Engine Tuning Advisor (Figure 12 and Figure 14):



```
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
```

-BILGISAYAR - QueryTraceFile

Estimated improvement: 86%

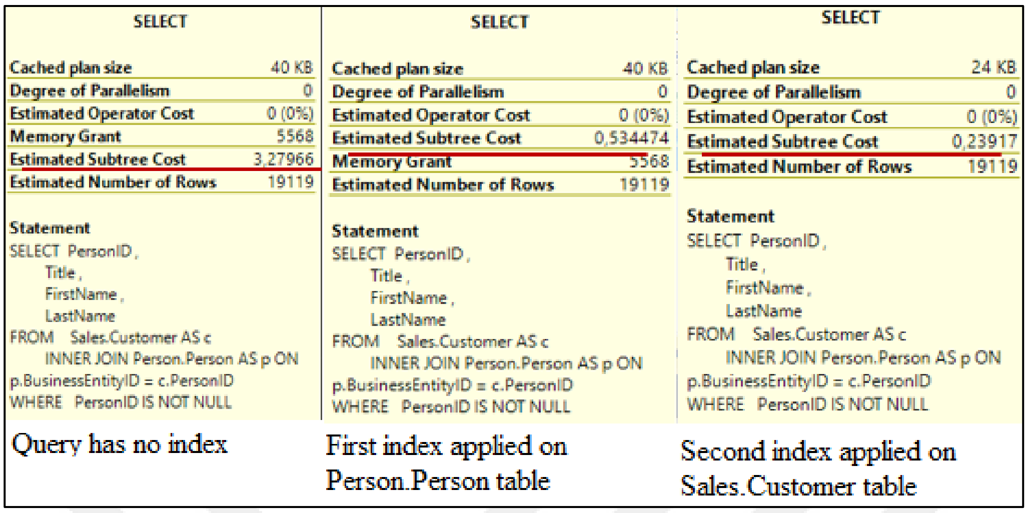
Index Recommendations

Database Name	Object Name	Recommendation	Target of Recommendation	Details	Partition Sch
AdventureWorks2014	[Person].[Person]	create	_dta_index_Person_5_1765581328__K1_4_5_7		
AdventureWorks2014	[Sales].[Customer]	create	_dta_index_Customer_5_997578592__K2		

Figure 12. Analyzing of Third Query.

After analyzing third query’s trace file Database Engine Tuning Advisor gave two indexes recommendation with 86% estimated performance improvement. After applied recommended indexes, users can control query performance from execution plan. With the help of execution plan’s Tooltip that is a yellow box, it shows up when the mouse is over the command. Execution Plan and Tooltip will be discussed in detail

Execution Plan section. For now to measure of improvement, it will be compared Estimated Subtree Cost values in the tooltip (Figure 13).



SELECT

Cached plan size	40 KB
Degree of Parallelism	0
Estimated Operator Cost	0 (0%)
Memory Grant	5568
Estimated Subtree Cost	3,27966
Estimated Number of Rows	19119

Statement
```
SELECT PersonID ,
       Title ,
       FirstName ,
       LastName
FROM   Sales.Customer AS c
       INNER JOIN Person.Person AS p ON
p.BusinessEntityID = c.PersonID
WHERE  PersonID IS NOT NULL
```

Query has no index

SELECT

Cached plan size	40 KB
Degree of Parallelism	0
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,534474
Memory Grant	5568
Estimated Number of Rows	19119

Statement
```
SELECT PersonID ,
       Title ,
       FirstName ,
       LastName
FROM   Sales.Customer AS c
       INNER JOIN Person.Person AS p ON
p.BusinessEntityID = c.PersonID
WHERE  PersonID IS NOT NULL
```

First index applied on Person.Person table

SELECT

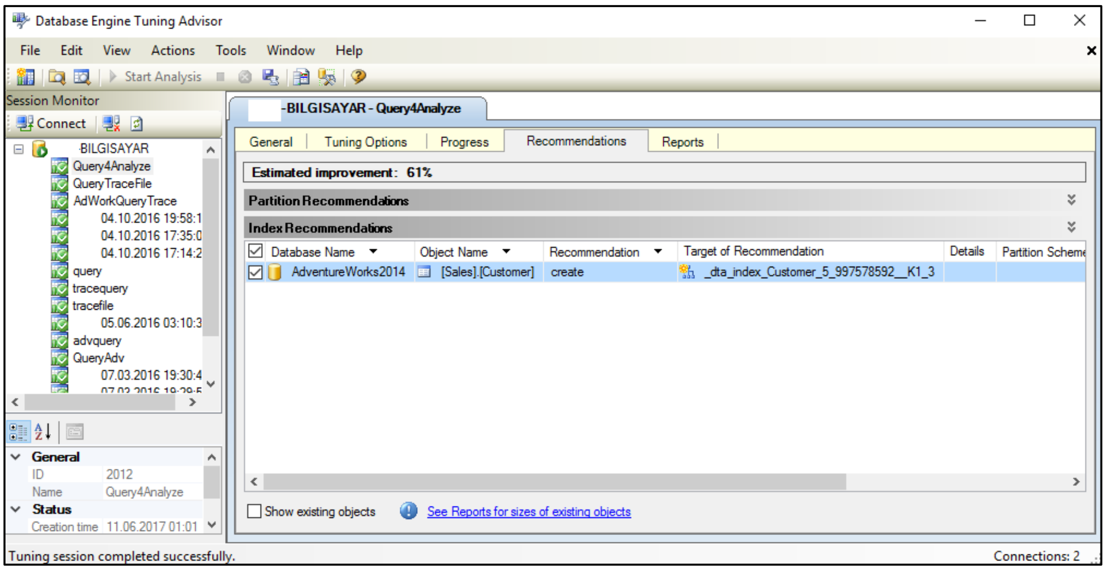
Cached plan size	24 KB
Degree of Parallelism	0
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,23917
Estimated Number of Rows	19119

Statement
```
SELECT PersonID ,
       Title ,
       FirstName ,
       LastName
FROM   Sales.Customer AS c
       INNER JOIN Person.Person AS p ON
p.BusinessEntityID = c.PersonID
WHERE  PersonID IS NOT NULL
```

Second index applied on Sales.Customer table

Figure 13. The Performance Impact of Index.

As shown in the Figure 13; if recommended indexes are not applied on third query, this query performance will be 3,27966. First index applied on person.person table and it increased performance 3,27966 to 0,534474. Then second index index applied on sales.customer table to see how it will affect performance, and it’s result is satisfying as first index, it increase query performance 0,534474 to 0,23917. So, it can be said that recommended indexes are useful for this query, as a result of Database Engine Tuning Advisor recommendation. After this query examination, fourth query improvement can be examined with same steps; primarily create trace file for fourth query then analyze this file with Database Engine Tuning Advisor and get some index recommendations (Figure 14).


Database Engine Tuning Advisor

File Edit View Actions Tools Window Help

Session Monitor

-BILGISAYAR - Query4Analyze

General | Tuning Options | Progress | Recommendations | Reports

Estimated improvement: 61%

Partition Recommendations

Index Recommendations

Database Name	Object Name	Recommendation	Target of Recommendation	Details	Partition Scheme
AdventureWorks2014	[Sales].[Customer]	create	_dta_index_Customer_5_997578592__K1_3		

Show existing objects

See Reports for sizes of existing objects

General: ID 2012; Name Query4Analyze

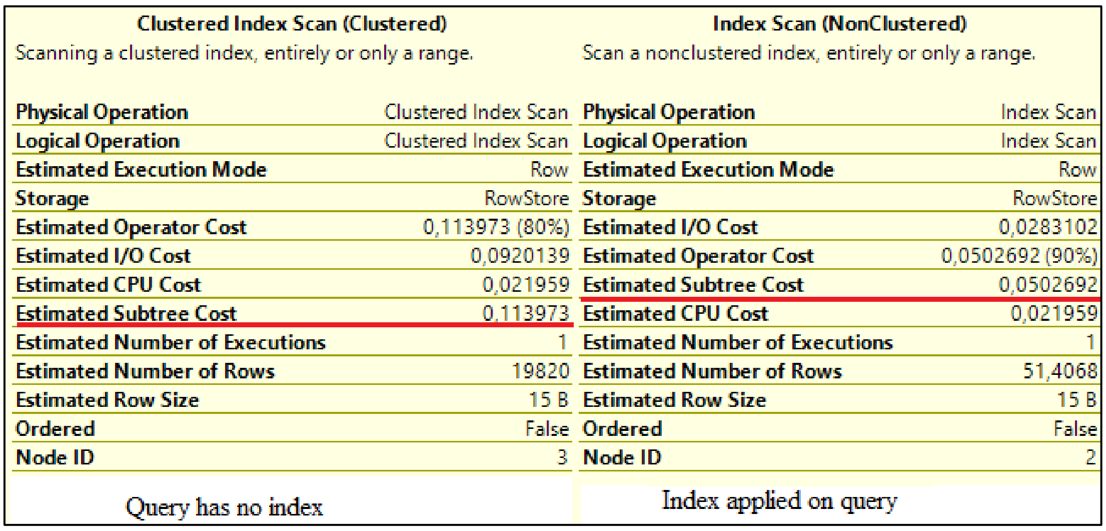
Status: Creation time 11.06.2017 01:01

Tuning session completed successfully.

Connections: 2

Figure 14. Index recommendation for Query Four.

Estimated improvement of query performance is 61%, so if recommended index is applied on the query, performance will increase 61%. After applied index, it will be seen the difference of performance between two tooltips of query four (Figure 15).



Clustered Index Scan (Clustered)		Index Scan (NonClustered)	
Scanning a clustered index, entirely or only a range.		Scan a nonclustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan	Physical Operation	Index Scan
Logical Operation	Clustered Index Scan	Logical Operation	Index Scan
Estimated Execution Mode	Row	Estimated Execution Mode	Row
Storage	RowStore	Storage	RowStore
Estimated Operator Cost	0,113973 (80%)	Estimated I/O Cost	0,0283102
Estimated I/O Cost	0,0920139	Estimated Operator Cost	0,0502692 (90%)
Estimated CPU Cost	0,021959	Estimated Subtree Cost	0,0502692
Estimated Subtree Cost	0,113973	Estimated CPU Cost	0,021959
Estimated Number of Executions	1	Estimated Number of Executions	1
Estimated Number of Rows	19820	Estimated Number of Rows	51,4068
Estimated Row Size	15 B	Estimated Row Size	15 B
Ordered	False	Ordered	False
Node ID	3	Node ID	2
Query has no index		Index applied on query	

Figure 15. The Performance Impact of Index.

As shown in the Figure 15, performance of fourth query increase 0,113973 to 0,0502692. Therefore, it can be said that performance improvement works have a positive effect.

2.3 ACTIVITY MONITOR

The SQL Server Activity Monitor is a tool in SQL Server that displays information about SQL Server processes and their effect on SQL Server performance (Petrovic, 2014b). Activity Monitor runs queries on the monitored instance to obtain information for the Activity Monitor display panes. When the refresh interval is set to less than 10 seconds, the time that is used to run these queries can affect server performance. Activity monitor consists of some panes: Overview, Processes, Resources Waits, Data file I/O and Recent Expensive Queries. The Recent Expensive Queries pane will be the focal point for this study.

i. The Overview Pane

This pane contains four graphs (Figure 16):

- *Processor Time:* The percentage of elapsed time that the processor spends to execute non-idle threads for the instance across all CPUs.
- *Waiting Tasks:* The number of tasks that are waiting for processor, I/O, or memory resources.
- *Database I/O:* Information on data and log files for system and user databases.
- *Batch Request/sec:* The number of SQL Server batches that are received by the instance.

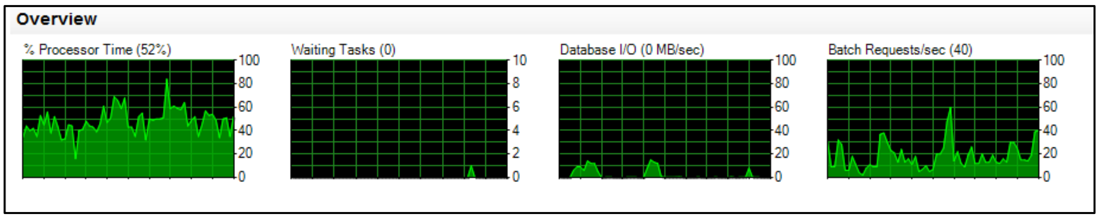
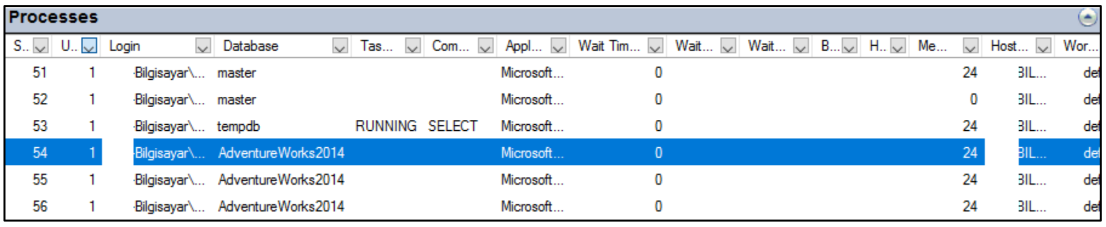


Figure 16. Display of Overview Pane (mssqldude.wordpress.com, 2011)

With the help of these graphs, Users can make the necessary arrangements.

ii. The Processes Pane

The Processes Pane shows the information about the currently running processes on the SQL databases, who runs them, and from which application (Figure 17) (Petrovicb, 2014).

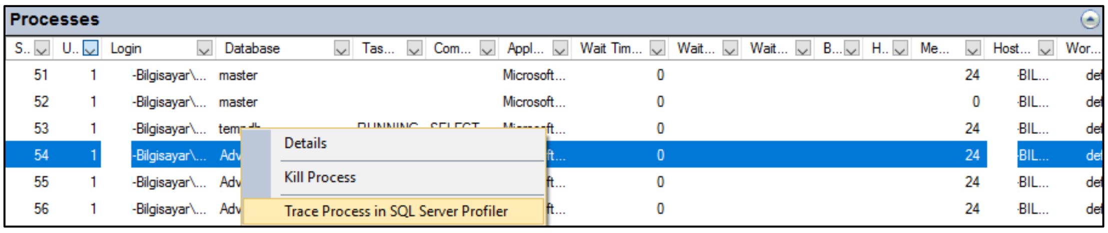


Processes

S..	U..	Login	Database	Tas...	Com...	Appl...	Wait Tim...	Wait...	Wait...	B...	H..	Me...	Host...	Wor...
51	1	Bilgisayar\...	master			Microsoft...	0					24	BIL...	de
52	1	Bilgisayar\...	master			Microsoft...	0					0	BIL...	de
53	1	Bilgisayar\...	tempdb	RUNNING	SELECT	Microsoft...	0					24	BIL...	de
54	1	Bilgisayar\...	AdventureWorks2014			Microsoft...	0					24	BIL...	de
55	1	Bilgisayar\...	AdventureWorks2014			Microsoft...	0					24	BIL...	de
56	1	Bilgisayar\...	AdventureWorks2014			Microsoft...	0					24	BIL...	de

Figure 17. Display of Process Pane.

Figure 17 shows the all the active users and running processes on the SQL Server. Users can right click any of the Session IDs that users think are problematic and can run a SQL Server Profiler Trace to capture all its activities; users can also see the Session Details or can even kill a process (Figure 18) (Mehta, 2010).



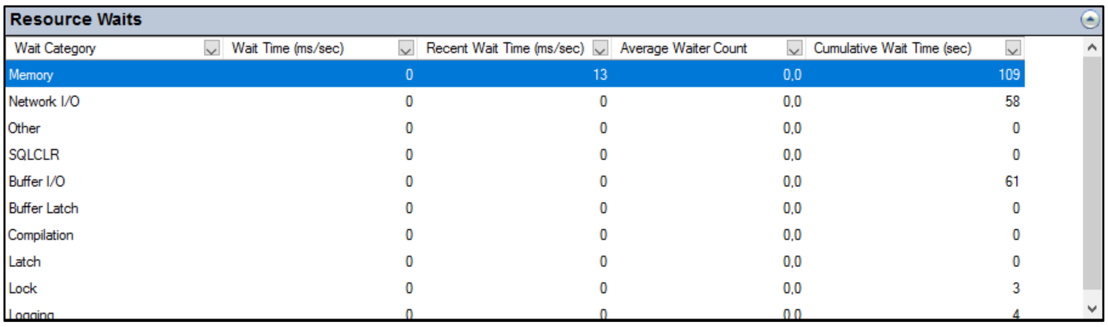
Processes

S..	U..	Login	Database	Tas...	Com...	Appl...	Wait Tim...	Wait...	Wait...	B...	H..	Me...	Host...	Wor...
51	1	Bilgisayar\...	master			Microsoft...	0					24	BIL...	de
52	1	Bilgisayar\...	master			Microsoft...	0					0	BIL...	de
53	1	Bilgisayar\...	temp			...ft...	0					24	BIL...	de
54	1	Bilgisayar\...	Adv			ft...	0					24	BIL...	de
55	1	Bilgisayar\...	Adv			ft...	0					24	BIL...	de
56	1	Bilgisayar\...	Adv			ft...	0					24	BIL...	de

Figure 18. Selections of Processes Pane.

iii. The Resources Waits

This Activity Monitor pane shows the information about waits for resources (Figure 19).



Resource Waits

Wait Category	Wait Time (ms/sec)	Recent Wait Time (ms/sec)	Average Waiter Count	Cumulative Wait Time (sec)
Memory	0	13	0,0	109
Network I/O	0	0	0,0	58
Other	0	0	0,0	0
SQLCLR	0	0	0,0	0
Buffer I/O	0	0	0,0	61
Buffer Latch	0	0	0,0	0
Compilation	0	0	0,0	0
Latch	0	0	0,0	0
Lock	0	0	0,0	3
Logging	0	0	0,0	4

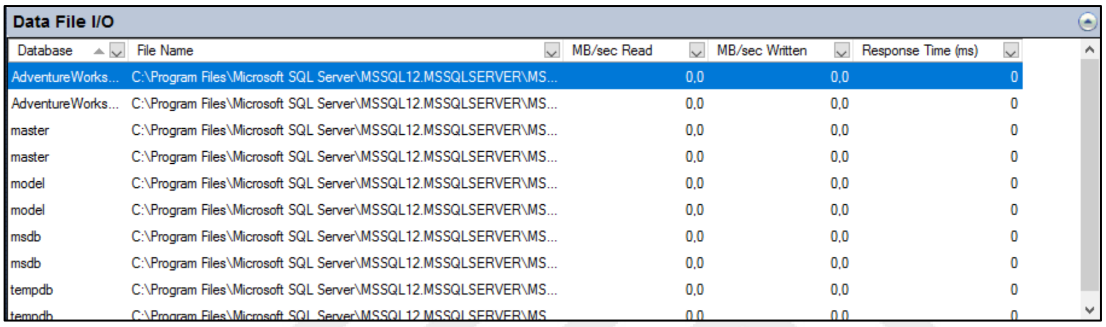
Figure 19. Display of Resources Waits Pane.

- Wait Category: Wait type of Process Pane,
- Wait Time (ms/sec): The time all waiting tasks are waiting for one or more resources,
- Recent Wait Time (ms/sec): The average time all waiting tasks are waiting for one or more resources,

- Average Waiter Count: Is calculated for a typical point in time in the last sample interval and represents the number of tasks waiting for one or more resources,
- Cumulative Wait Time (sec): The total time waiting tasks have waited for one or more resources since the last SQL Server restart, or DBCC SQLPERF last execution (Petrovic, 2014b).

iv. The Data File I/O Pane

Contains information about the database files, all database files are listed (Figure 20).



Data File I/O

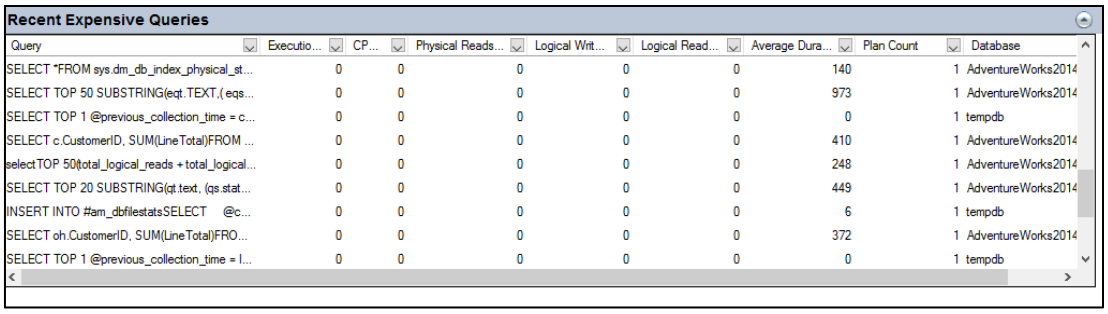
Database	File Name	MB/sec Read	MB/sec Written	Response Time (ms)
AdventureWorks...	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
AdventureWorks...	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
master	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
master	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
model	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
model	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
msdb	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
msdb	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
tempdb	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0
tempdb	C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MS...	0,0	0,0	0

Figure 20. Display of Data File I/O.

- MB/sec Read: Shows recent read activity,
- MB/sec Written: Shows recent write activity,
- Response Time (ms): Average response time for recent read-and-write activity.

v. The Recent Expensive Queries Pane

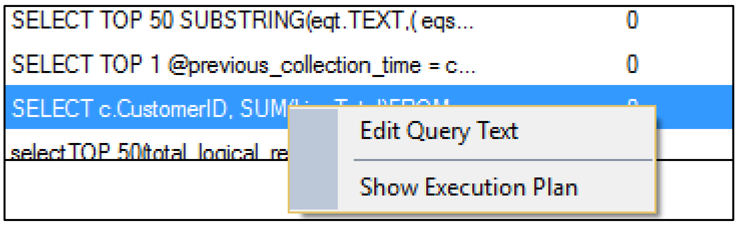
This pane is more useful to find the most expensive queries that means these queries consume much more time than others do. For executed queries, users get list the list shows expensive queries executed in the last 30 seconds. Users can control and edit these queries. In addition, Execution Plan can be displayed and to see options right click can be used (Figure 21 and Figure 22):



Recent Expensive Queries

Query	Executio...	CP...	Physical Reads...	Logical Writ...	Logical Read...	Average Dura...	Plan Count	Database
SELECT *FROM sys.dm_db_index_physical_st...	0	0	0	0	0	140	1	AdventureWorks2014
SELECT TOP 50 SUBSTRING(eqt.TEXT,(eqs...	0	0	0	0	0	973	1	AdventureWorks2014
SELECT TOP 1 @previous_collection_time = c...	0	0	0	0	0	0	1	tempdb
SELECT c.CustomerID, SUM(LineTotal)FROM ...	0	0	0	0	0	410	1	AdventureWorks2014
select TOP 50(total_logical_reads + total_logical...	0	0	0	0	0	248	1	AdventureWorks2014
SELECT TOP 20 SUBSTRING(qt.text, (qs.stat...	0	0	0	0	0	449	1	AdventureWorks2014
INSERT INTO #am_dbfilestatsSELECT @c...	0	0	0	0	0	6	1	tempdb
SELECT oh.CustomerID, SUM(LineTotal)FRO...	0	0	0	0	0	372	1	AdventureWorks2014
SELECT TOP 1 @previous_collection_time = l...	0	0	0	0	0	0	1	tempdb

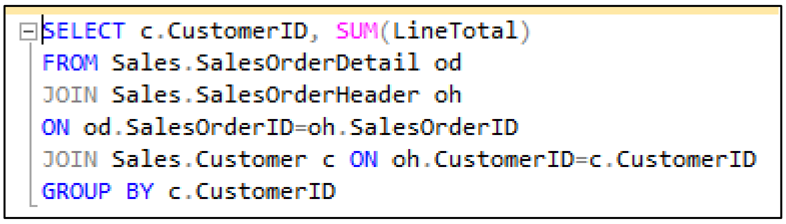
Figure 21. Display of Recent Expensive Queries Pane.



```
SELECT TOP 50 SUBSTRING(eqt.TEXT,( eqs...        0
SELECT TOP 1 @previous_collection_time = c...    0
SELECT c.CustomerID, SUM(
select TOP 50(total_logical_re
```

Figure 22. Right Click Options of Expensive Query.

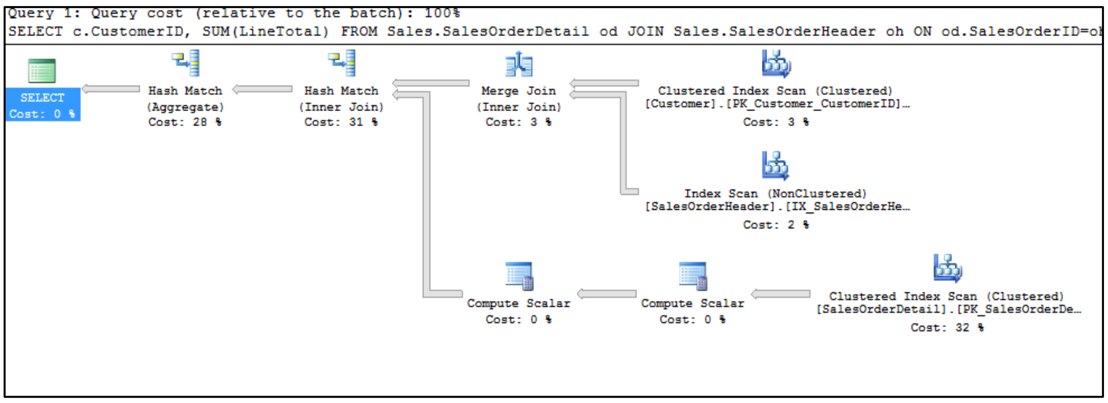
Query can be edited directly; different query can written for same reseult or can be idenfied an index for this query (Figure 23).



```
SELECT c.CustomerID, SUM(LineTotal)
FROM Sales.SalesOrderDetail od
JOIN Sales.SalesOrderHeader oh
ON od.SalesOrderID=oh.SalesOrderID
JOIN Sales.Customer c ON oh.CustomerID=c.CustomerID
GROUP BY c.CustomerID
```

Figure 23. Edit Query Selection.

Or users can reach execution plan of this expensive query; they can see cost of query and get more information using with tooltips (Figure 24).



```
Query 1: Query cost (relative to the batch): 100%
SELECT c.CustomerID, SUM(LineTotal) FROM Sales.SalesOrderDetail od JOIN Sales.SalesOrderHeader oh ON od.SalesOrderID=oh
```

SELECT
Cost: 0 %
Hash Match
(Aggregate)
Cost: 28 %
Hash Match
(Inner Join)
Cost: 31 %
Merge Join
(Inner Join)
Cost: 3 %
Clustered Index Scan (Clustered)
[Customer].[PK_Customer_CustomerID]…
Cost: 3 %
Index Scan (NonClustered)
[SalesOrderHeader].[IX_SalesOrderHe…
Cost: 2 %
Compute Scalar
Cost: 0 %
Compute Scalar
Cost: 0 %
Clustered Index Scan (Clustered)
[SalesOrderDetail].[PK_SalesOrderDe…
Cost: 32 %

Figure 24. Execution Plan Selection.

2.4 EXECUTION PLAN

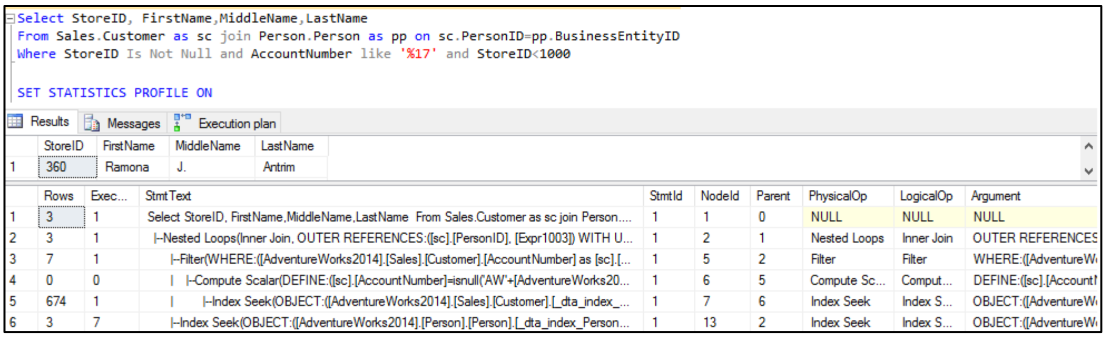
Execution Plan is the most useful feature of SQL Server to measure, monitor and optimize expensive queries. Execution plan is a graph that shows path of the query execution. To read it, users should start reading right to left and top to bottom. Especially it can be used to find query performance problem, firstly problematic queries found then their Execution Plan examined to find problem. Execution plans stored at cache and if same query runs again this plan is used. Some changings effects execution plan so there is some differences will be occurred on execution plan. This changings can be (Adar, 2016):

- Table changing,
- DROP or CREATE index command,
- Changing of STATISTICS,
- Calling sp_recompile function,
- Using DDL and DML together.

Execution Plan has three different type; Graphical Plan, Text Plan and XML Plan. In this study graphical plan is examined deeply.

2.4.1 Text Plan:

Reading text plan of executed query is hard and complex for users. To use text form of execution plan users should execute; SET STATISTICS PROFILE ON command for display Actual Execution Text Plan and SET SHOWPLAN_ALL ON command for Estimated Execution Text Plan (Figure 25).



```sql
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000

SET STATISTICS PROFILE ON
```

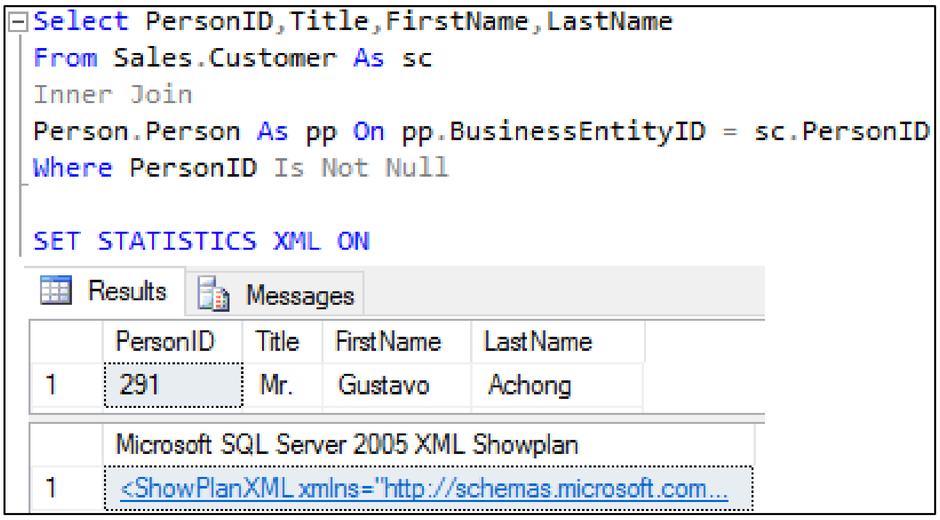
	StoreID	FirstName	MiddleName	LastName
1	360	Ramona	J.	Antrim

	Rows	Exec...	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument
1	3	1	Select StoreID, FirstName,MiddleName,LastName From Sales.Customer as sc join Person....	1	1	0	NULL	NULL	NULL
2	3	1	|--Nested Loops(Inner Join, OUTER REFERENCES:([sc].[PersonID], [Expr1003]) WITH U...	1	2	1	Nested Loops	Inner Join	OUTER REFERENCES
3	7	1	|--Filter(WHERE:([AdventureWorks2014].[Sales].[Customer].[AccountNumber] as [sc].[...	1	5	2	Filter	Filter	WHERE:([AdventureW
4	0	0	| |--Compute Scalar(DEFINE:([sc].[AccountNumber]=isnull('AW'+[AdventureWorks20...	1	6	5	Compute Sc...	Comput...	DEFINE:([sc].[Account
5	674	1	| |--Index Seek(OBJECT:([AdventureWorks2014].[Sales].[Customer].[_dta_index_...	1	7	6	Index Seek	Index S...	OBJECT:([AdventureW
6	3	7	|--Index Seek(OBJECT:([AdventureWorks2014].[Person].[Person].[_dta_index_Person...	1	13	2	Index Seek	Index S...	OBJECT:([AdventureW

Figure 25. Display of Text Plan.

2.4.2 XML Plan

Reading XML form of execution plan is easier than Text plan form. Users should display XML form as a graph. To use xml form of execution plan users should execute; SET STATISTICS XML ON command for display Actual XML Plan and SET SHOWPLAN_XML ON command for Estimated Execution XML Plan (Figure 26 and Figure 27).

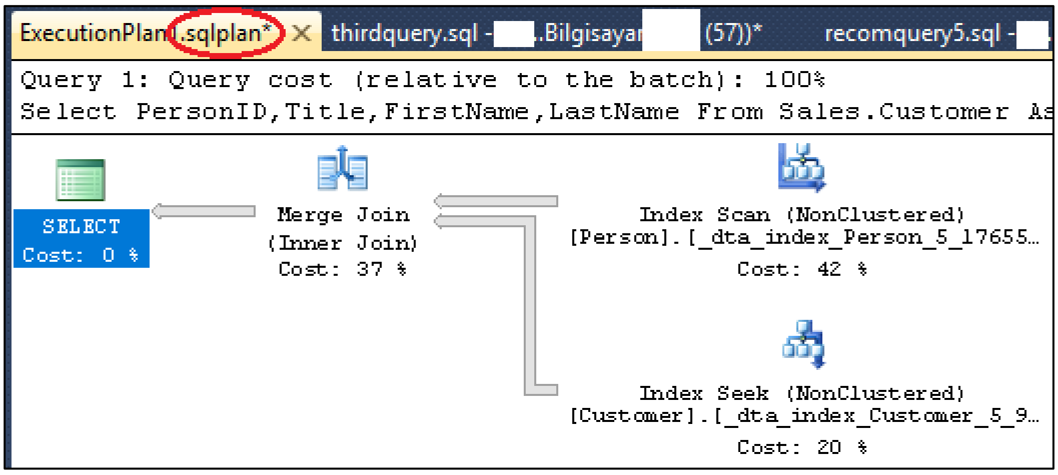


```sql
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null

SET STATISTICS XML ON
```

	PersonID	Title	FirstName	LastName
1	291	Mr.	Gustavo	Achong

	Microsoft SQL Server 2005 XML Showplan
1	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Figure 26. XML Plan File.


ExecutionPlan[redacted].sqlplan*
thirdquery.sql - [redacted].Bilgisayar[redacted] (57))*
recomquery5.sql - [redacted]

```
Query 1: Query cost (relative to the batch): 100%
Select PersonID,Title,FirstName,LastName From Sales.Customer As
```

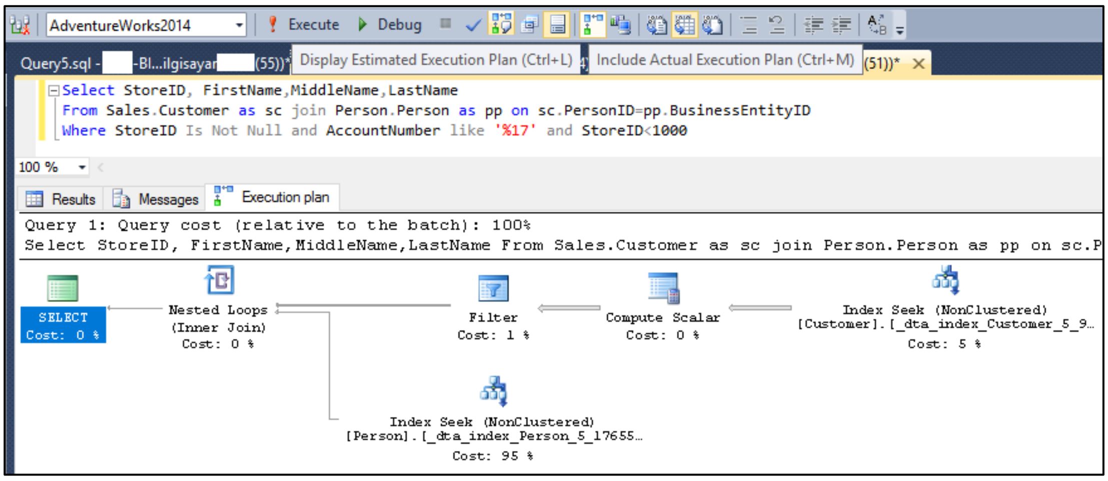
SELECT
Cost: 0 %
Merge Join
(Inner Join)
Cost: 37 %
Index Scan (NonClustered)
[Person].[_dta_index_Person_5_17655…
Cost: 42 %
Index Seek (NonClustered)
[Customer].[_dta_index_Customer_5_9…
Cost: 20 %

Figure 27. Display XML Plan as a Graph.

2.4.3 Graphical Plan

Graphical Plan is the most useful and understandable Execution Plan form. Because graphs show all necessary information about query operators and their tooltips that have all information about operators. With using this information, users can find the problem of their query. Query plan graphs can be displayed as Estimated Execution Plan or Actual Execution Plan. Estimated execution plan represent an estimated query plan that created from query optimizer before query execution. Nevertheless, actual execution plan represent real plan that occurs after executed query.

Users can select execution plan form from the top of SQL Server than execute the query (Figure 28):


AdventureWorks2014
Execute
Debug
Query5.sql - [redacted]-Bl...ilgisayar[redacted](55))*
Display Estimated Execution Plan (Ctrl+L)
Include Actual Execution Plan (Ctrl+M)
(51))*

```
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
```

100 %
Results
Messages
Execution plan

```
Query 1: Query cost (relative to the batch): 100%
Select StoreID, FirstName,MiddleName,LastName From Sales.Customer as sc join Person.Person as pp on sc.P
```

SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
Filter
Cost: 1 %
Compute Scalar
Cost: 0 %
Index Seek (NonClustered)
[Customer].[_dta_index_Customer_5_9…
Cost: 5 %
Index Seek (NonClustered)
[Person].[_dta_index_Person_5_17655…
Cost: 95 %

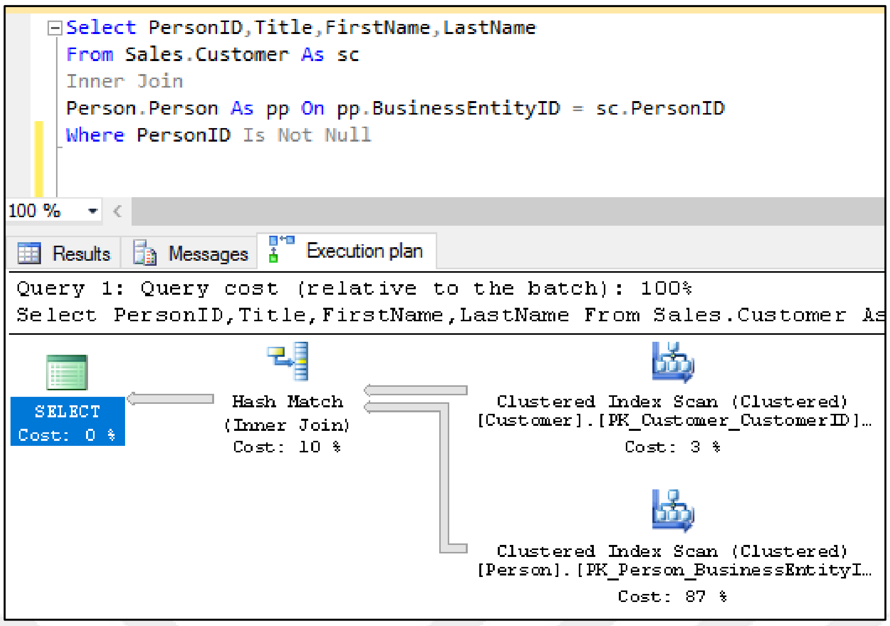
Figure 28. Execute Query with Execution Plan Selection.

As is seen in the Figure 28, query execute with execution plan selection, SQL Server displays graphical plan of query execution plan. With using this plan, users can check costs of operators and which index operations (index seek or index scan) are used for finding result. If there is index scan or table scan, it means it should be started here to optimize this query. Because, scanning all data to find the data that the query wants, is a time wasting and performance killer. So, seek operator is the preferable for quick result and performance improving.

The other advantage is tooltips, each icon on the graphical plan has an own tooltip. It can be said that these tooltips are lifesaving, because they have a lot of information about icons if users read right this given informations they can make an improvement on expensive queries. Two different queries will be used to show different tooltip contents with using execution plan tooltips.

As stated from Activity Monitor part, users can find expensive queries that means these queries consume much more time to execute than others, so Recent Expensive Queries Pane gives users them. After identifying expensive query users should examine their execution plan graph to find any problematic look.

Initially, fourth query will be examined in Figure 29; graph shows clustered index scan is used for bring result and server spends 87% of time. At first appearance, users think that index should be used for this query but before apply index, tooltips must be checked first. To make sure how intervention will be applied on query. Thus they have to know the meanings of expressions of tooltips (Figure 30).



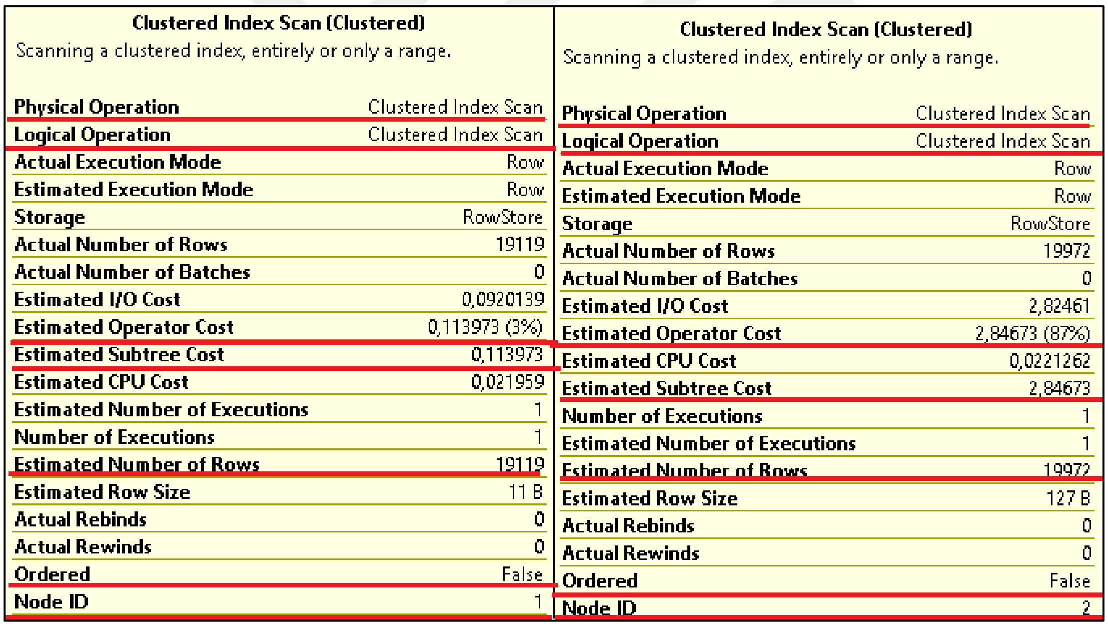
```
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
```

100 %
Results
Messages
Execution plan

```
Query 1: Query cost (relative to the batch): 100%
Select PersonID,Title,FirstName,LastName From Sales.Customer As
```

SELECT
Cost: 0 %
Hash Match
(Inner Join)
Cost: 10 %
Clustered Index Scan (Clustered)
[Customer].[PK_Customer_CustomerID]…
Cost: 3 %
Clustered Index Scan (Clustered)
[Person].[PK_Person_BusinessEntityI…
Cost: 87 %

Figure 29. Execution Plan of Fourth Query.



Clustered Index Scan (Clustered)
Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	19119
Actual Number of Batches	0
Estimated I/O Cost	0,0920139
Estimated Operator Cost	0,113973 (3%)
Estimated Subtree Cost	0,113973
Estimated CPU Cost	0,021959
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	19119
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	1

Clustered Index Scan (Clustered)
Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	19972
Actual Number of Batches	0
Estimated I/O Cost	2,82461
Estimated Operator Cost	2,84673 (87%)
Estimated CPU Cost	0,0221262
Estimated Subtree Cost	2,84673
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	19972
Estimated Row Size	127 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	2

Figure 30. Display of Icons' Tooltips.

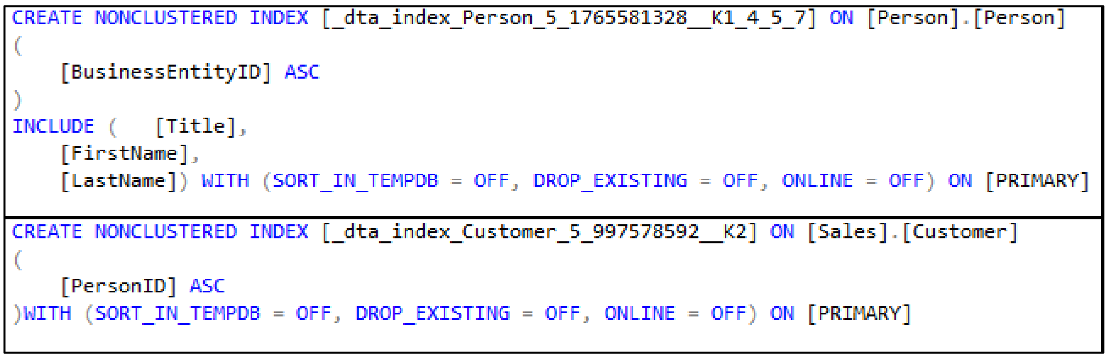
Some important expressions can be explained;

- Physical Operation shows the actual operation when the query is executed.
- Logical Operation shows the estimated operation when the query is executed.
- Estimated Operator Cost shows the all-time that spends on this operation.
- Estimated Subtree Cost shows the total time the SQL Server spent.

- Estimated Number of Rows shows The number of affected records.
- Ordered shows if the query has order by operation TRUE, has not FALSE. It should not be used if query is not too complex.
- Node ID transforms the execution plan into a tree structure and displays the level of the corresponding node.
- Index Scan means that all the data is read line by line.
- Index Seek means that an index that is clustered or non-clustered is detected and only the desired data is reached (Adar, 2016).

As stated previous Database Engine Tuning Advisor part, estimated subtree cost as a measure of improvement. In Figure 29 the query operation is Clustered Index Scan, effected rows are 19972 and Estimated Subtree Cost is 2,84673 to improve this query; query operation should be index seek, effected rows should be decreased and Estimated Subtree Cost has a lower value.

If users cannot decide which index will apply on the query, they can use Database Engine Tuning Advisor to get some recommendation like this query. After applying recommended indexes and it observed in the previous Database Engine Tuning Advisor section that improvement of performance with using SELECT operator tooltips to see all improvement. However, in this section it will be observed how indexes effect other index scan operators (Figure 31).



```
CREATE NONCLUSTERED INDEX [_dta_index_Person_5_1765581328__K1_4_5_7] ON [Person].[Person]
(
    [BusinessEntityID] ASC
)
INCLUDE (   [Title],
    [FirstName],
    [LastName]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
```

```
CREATE NONCLUSTERED INDEX [_dta_index_Customer_5_997578592__K2] ON [Sales].[Customer]
(
    [PersonID] ASC
)WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
```

Figure 31. Recommended Indexes.

The first intervention for Person.Person table because its cost has higher value, so users can apply index on this table like Figure 31. After applying non-clustered index on Person.Person table (Figure 32):

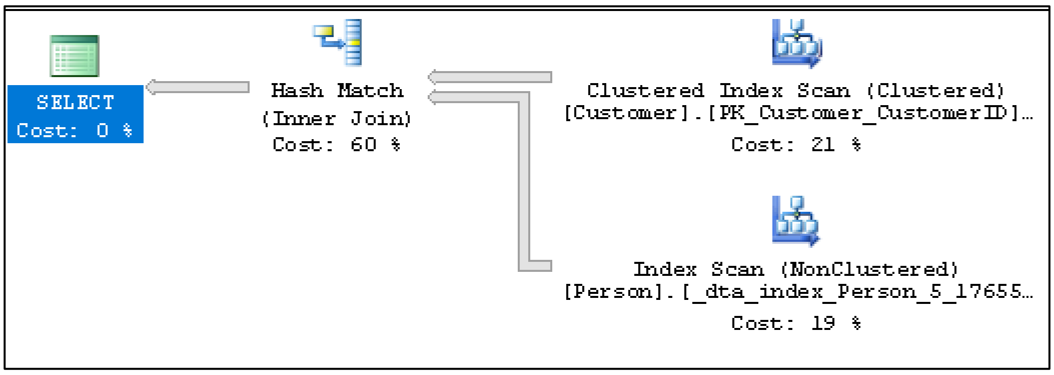
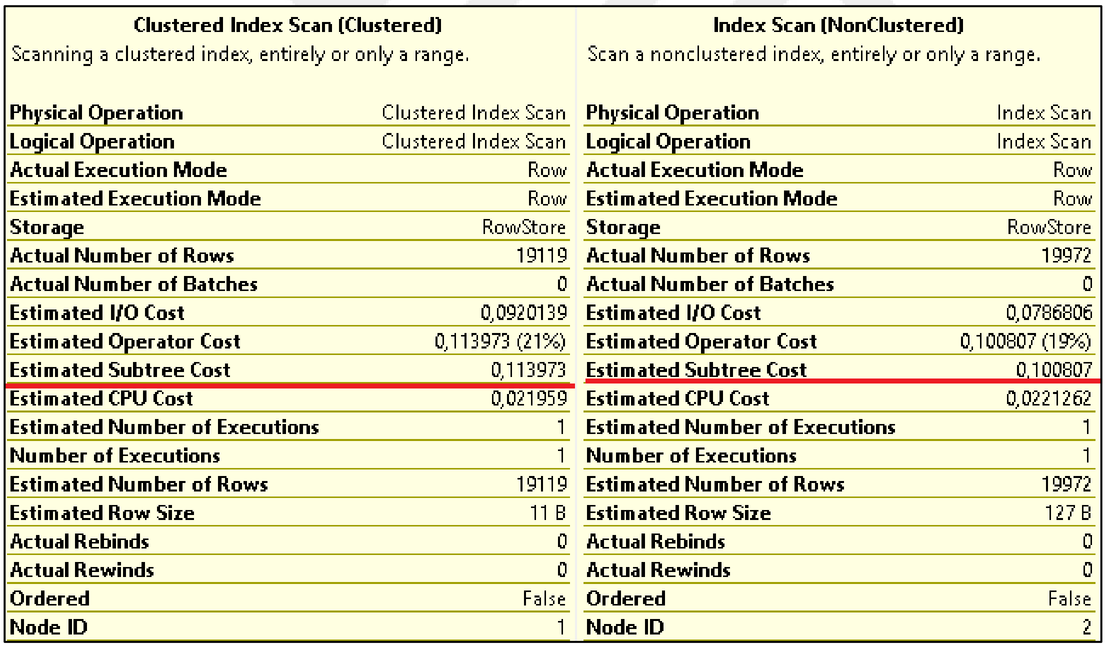

SELECT
Cost: 0 %
Hash Match
(Inner Join)
Cost: 60 %
Clustered Index Scan (Clustered)
[Customer].[PK_Customer_CustomerID]…
Cost: 21 %
Index Scan (NonClustered)
[Person].[_dta_index_Person_5_17655…
Cost: 19 %

Figure 32. Non-clustered index applied Person.Person table.

Cost is decreased by using index, person.person table does index scan again but tooltips should be examined before. And there is a new complication on sales.customer table as is seen, using index increase its cost, so this table needs an another index. Can be looked at the Figure 33 to understand difference of tooltips:



Clustered Index Scan (Clustered) Scanning a clustered index, entirely or only a range.		**Index Scan (NonClustered)** Scan a nonclustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan	**Physical Operation**	Index Scan
Logical Operation	Clustered Index Scan	**Logical Operation**	Index Scan
Actual Execution Mode	Row	**Actual Execution Mode**	Row
Estimated Execution Mode	Row	**Estimated Execution Mode**	Row
Storage	RowStore	**Storage**	RowStore
Actual Number of Rows	19119	**Actual Number of Rows**	19972
Actual Number of Batches	0	**Actual Number of Batches**	0
Estimated I/O Cost	0,0920139	**Estimated I/O Cost**	0,0786806
Estimated Operator Cost	0,113973 (21%)	**Estimated Operator Cost**	0,100807 (19%)
Estimated Subtree Cost	0,113973	**Estimated Subtree Cost**	0,100807
Estimated CPU Cost	0,021959	**Estimated CPU Cost**	0,0221262
Estimated Number of Executions	1	**Estimated Number of Executions**	1
Number of Executions	1	**Number of Executions**	1
Estimated Number of Rows	19119	**Estimated Number of Rows**	19972
Estimated Row Size	11 B	**Estimated Row Size**	127 B
Actual Rebinds	0	**Actual Rebinds**	0
Actual Rewinds	0	**Actual Rewinds**	0
Ordered	False	**Ordered**	False
Node ID	1	**Node ID**	2

Figure 33. Tooltips After First Index on Person.person table.

There is not difference on clustered index scan operator. However, in index scan operator estimated subtree cost has a huge difference. It can be said that second index can be applied on sales.customer table (Figure 34).

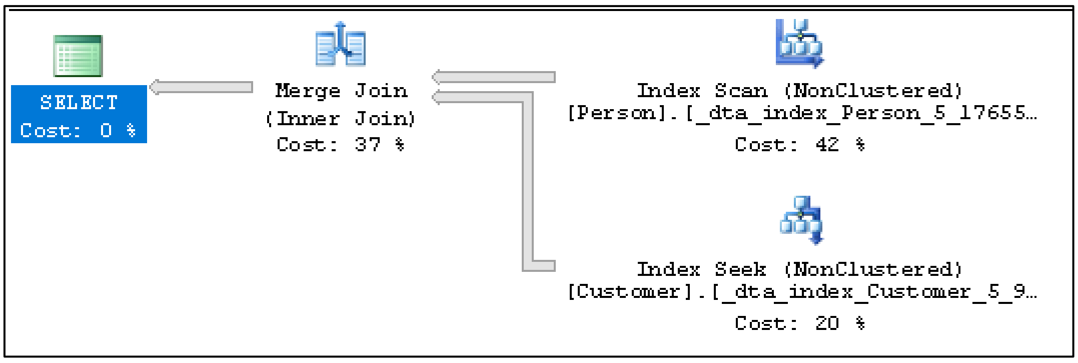
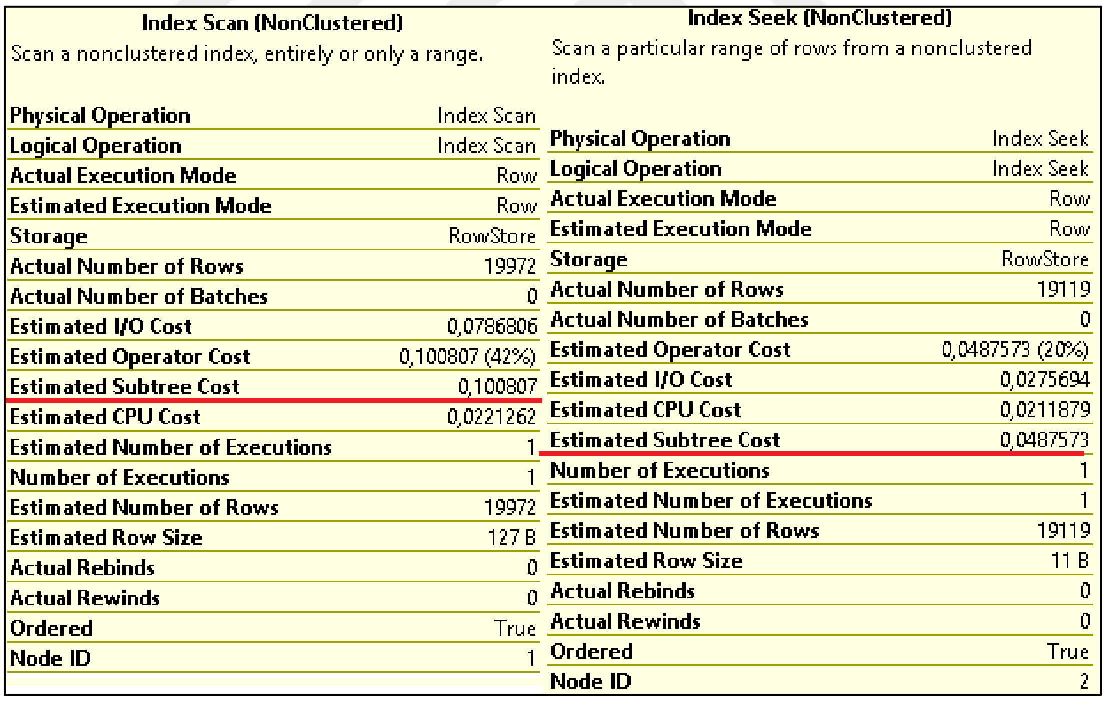

SELECT
Cost: 0 %
Merge Join
(Inner Join)
Cost: 37 %
Index Scan (NonClustered)
[Person].[_dta_index_Person_5_17655...
Cost: 42 %
Index Seek (NonClustered)
[Customer].[_dta_index_Customer_5_9...
Cost: 20 %

Figure 34. Non-clustered index applied Sales.Customer table.

After applying non-clustered index on sales.customer table person.person table does index scan with 42% cost. But, in sales.customer table does index seek with 20% cost. With this schema on Figure 34 users may think these indexes are not useful for improvement but if they check tooltips; they will see the difference and improvement of performance.



Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	19972
Actual Number of Batches	0
Estimated I/O Cost	0,0786806
Estimated Operator Cost	0,100807 (42%)
Estimated Subtree Cost	0,100807
Estimated CPU Cost	0,0221262
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	19972
Estimated Row Size	127 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	1

Index Seek (NonClustered)

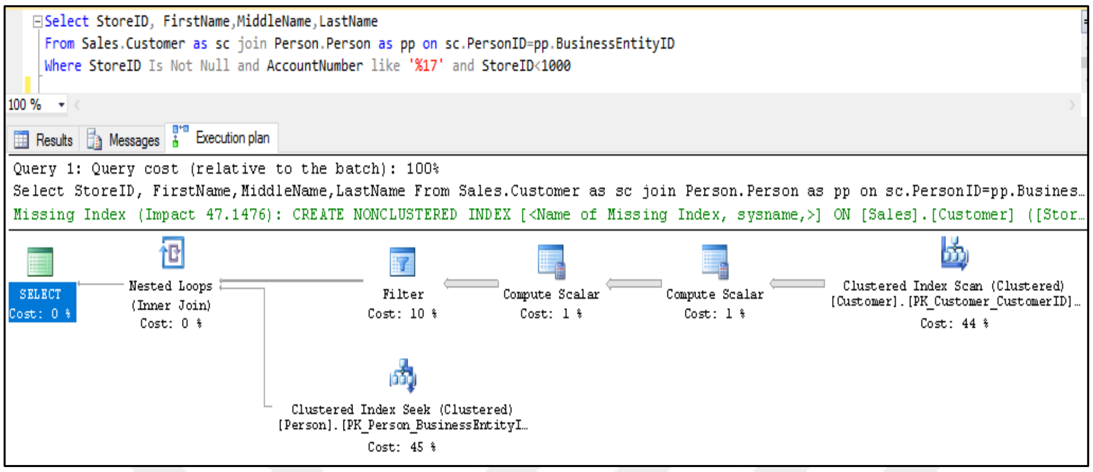
Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	19119
Actual Number of Batches	0
Estimated Operator Cost	0,0487573 (20%)
Estimated I/O Cost	0,0275694
Estimated CPU Cost	0,0211879
Estimated Subtree Cost	0,0487573
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	19119
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	2

Figure 35. Tooltips after Second Applied Index on Sales.Customer table.

Both two indexes applied on query and there is really big difference about performance. So it can be said that the query performance is improved, person.person table estimated subtree cost was 2,84673 now 0,100807 and sales.customer table cost was 0,113973 now 0,0487573.

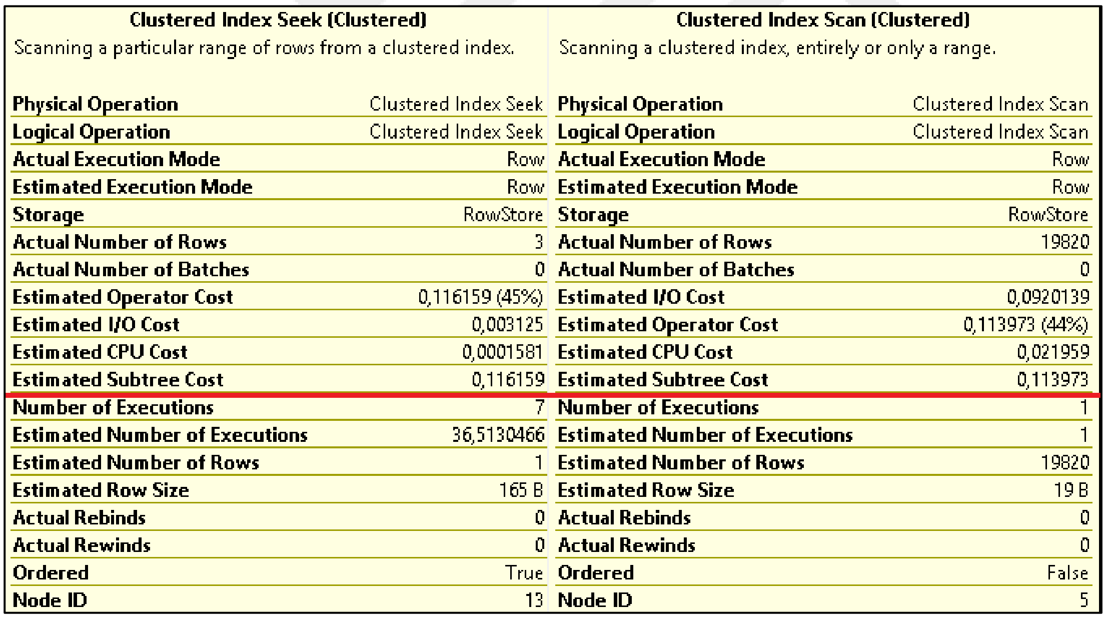
Fifth query examined with same way (Figure 36 and Figure 37):



```sql
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
```

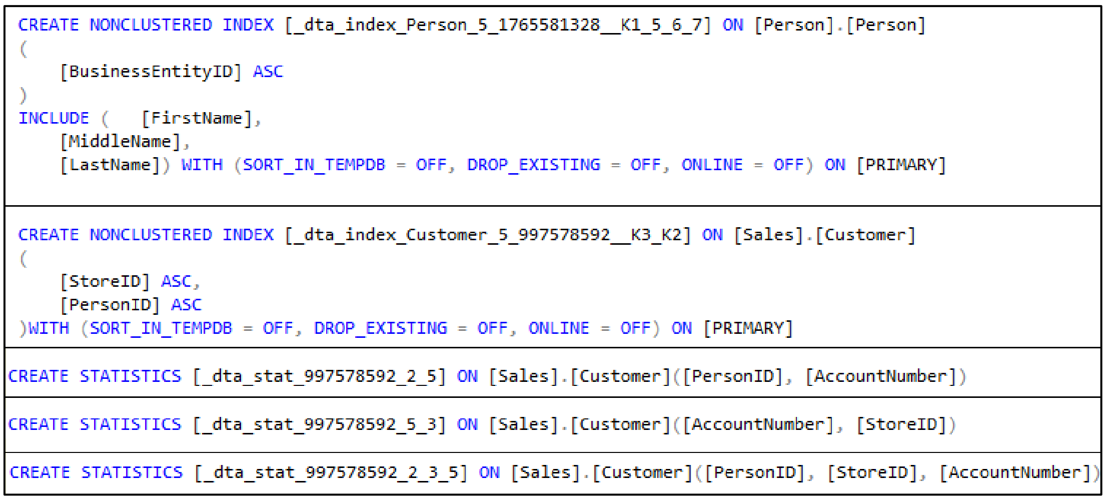
```
Query 1: Query cost (relative to the batch): 100%
Select StoreID, FirstName,MiddleName,LastName From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.Busines…
Missing Index (Impact 47.1476): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [Sales].[Customer] ([Stor…
```

Figure 36. Execution Plan of Fifth Query.



Clustered Index Seek (Clustered)		**Clustered Index Scan (Clustered)**	
Scanning a particular range of rows from a clustered index.		Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Seek	**Physical Operation**	Clustered Index Scan
Logical Operation	Clustered Index Seek	**Logical Operation**	Clustered Index Scan
Actual Execution Mode	Row	**Actual Execution Mode**	Row
Estimated Execution Mode	Row	**Estimated Execution Mode**	Row
Storage	RowStore	**Storage**	RowStore
Actual Number of Rows	3	**Actual Number of Rows**	19820
Actual Number of Batches	0	**Actual Number of Batches**	0
Estimated Operator Cost	0,116159 (45%)	**Estimated I/O Cost**	0,0920139
Estimated I/O Cost	0,003125	**Estimated Operator Cost**	0,113973 (44%)
Estimated CPU Cost	0,0001581	**Estimated CPU Cost**	0,021959
Estimated Subtree Cost	0,116159	**Estimated Subtree Cost**	0,113973
Number of Executions	7	**Number of Executions**	1
Estimated Number of Executions	36,5130466	**Estimated Number of Executions**	1
Estimated Number of Rows	1	**Estimated Number of Rows**	19820
Estimated Row Size	165 B	**Estimated Row Size**	19 B
Actual Rebinds	0	**Actual Rebinds**	0
Actual Rewinds	0	**Actual Rewinds**	0
Ordered	True	**Ordered**	False
Node ID	13	**Node ID**	5

Figure 37. Tooltips of Operators of Query five.



```
CREATE NONCLUSTERED INDEX [_dta_index_Person_5_1765581328__K1_5_6_7] ON [Person].[Person]
(
    [BusinessEntityID] ASC
)
INCLUDE (   [FirstName],
    [MiddleName],
    [LastName]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]

CREATE NONCLUSTERED INDEX [_dta_index_Customer_5_997578592__K3_K2] ON [Sales].[Customer]
(
    [StoreID] ASC,
    [PersonID] ASC
)WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]

CREATE STATISTICS [_dta_stat_997578592_2_5] ON [Sales].[Customer]([PersonID], [AccountNumber])

CREATE STATISTICS [_dta_stat_997578592_5_3] ON [Sales].[Customer]([AccountNumber], [StoreID])

CREATE STATISTICS [_dta_stat_997578592_2_3_5] ON [Sales].[Customer]([PersonID], [StoreID], [AccountNumber])
```

Figure 38. Recommendations of Data Base Engine Tuning Advisor.

For this query estimated subtree costs do not have very high value but yet Database Engine Tuning Advisor is used to see recommended index and improvement (Figure 38). Database Engine Tuning Advisor recommendations shown in the Figure 38. Three different statistics are recommended for improvement of performance on sales.customer table that means statistics also effect query performance so all recommendations applied on this query (Figure 39).

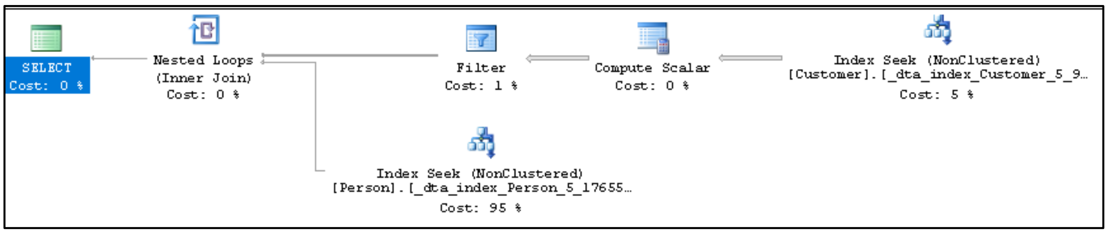
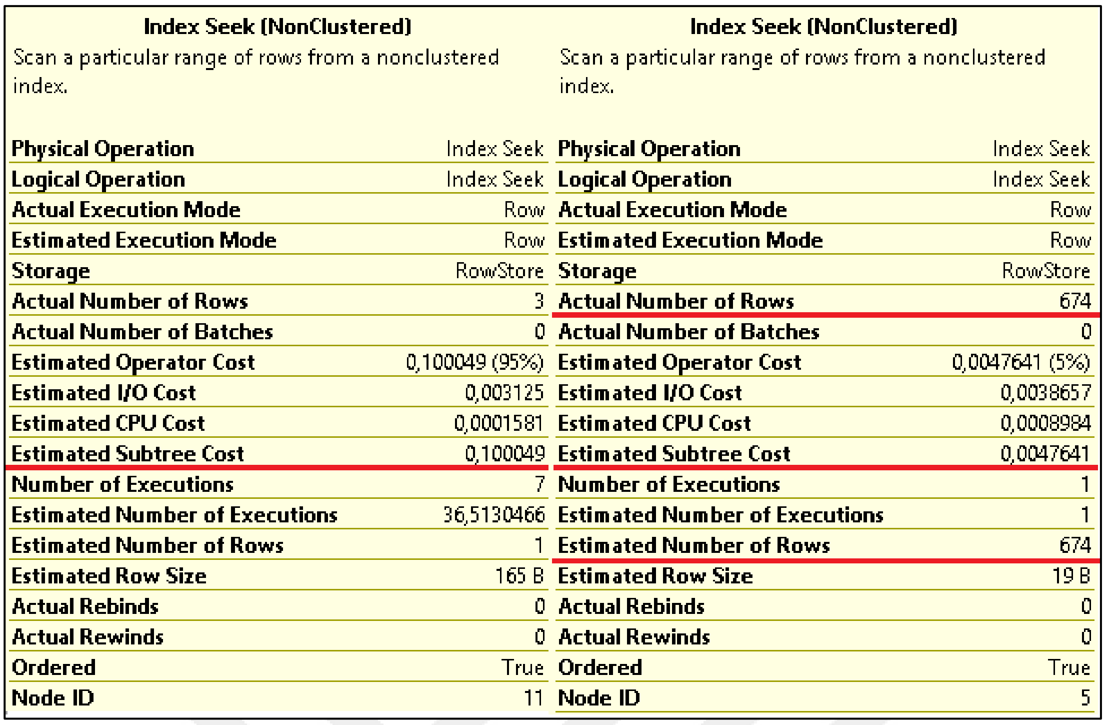

SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
Filter
Cost: 1 %
Compute Scalar
Cost: 0 %
Index Seek (NonClustered)
[Customer].[_dta_index_Customer_5_9...
Cost: 5 %
Index Seek (NonClustered)
[Person].[_dta_index_Person_5_17655...
Cost: 95 %

Figure 39. All Recommendations are Applied on The Query.



Index Seek (NonClustered)		Index Seek (NonClustered)	
Scan a particular range of rows from a nonclustered index.		Scan a particular range of rows from a nonclustered index.	
Physical Operation	Index Seek	Physical Operation	Index Seek
Logical Operation	Index Seek	Logical Operation	Index Seek
Actual Execution Mode	Row	Actual Execution Mode	Row
Estimated Execution Mode	Row	Estimated Execution Mode	Row
Storage	RowStore	Storage	RowStore
Actual Number of Rows	3	Actual Number of Rows	674
Actual Number of Batches	0	Actual Number of Batches	0
Estimated Operator Cost	0,100049 (95%)	Estimated Operator Cost	0,0047641 (5%)
Estimated I/O Cost	0,003125	Estimated I/O Cost	0,0038657
Estimated CPU Cost	0,0001581	Estimated CPU Cost	0,0008984
Estimated Subtree Cost	0,100049	Estimated Subtree Cost	0,0047641
Number of Executions	7	Number of Executions	1
Estimated Number of Executions	36,5130466	Estimated Number of Executions	1
Estimated Number of Rows	1	Estimated Number of Rows	674
Estimated Row Size	165 B	Estimated Row Size	19 B
Actual Rebinds	0	Actual Rebinds	0
Actual Rewinds	0	Actual Rewinds	0
Ordered	True	Ordered	True
Node ID	11	Node ID	5

Figure 40. Tooltips of Operators.

In the new execution plan there are two index seek with 5% and 95% costs. Users need to examine tooltips to measure performance changing (Figure 40). After applying all recommended command, the difference can be seen between Figure 37 and Figure 40; person.person table estimated subtree cost was 0,116159 now 0,100049 and sales.customer table cost was 0,113973 now 0,0047641 and with index seek on sales.customer table effected row numbers decreased 19820 to 674. At the result of this example, it can be said that query performance is improved.

CHAPTER THREE
EXAMINATON OF QUERIES WITH RELATIONAL ALGEBRA

Relational algebra is the basis of the SQL to understand query execution steps it must to known. Queries in relational algebra are applied to relation instances, result of a query is again a relation instance. It is useful for query execution it describes queries in the relational database. It can said relational algebra is a mathematical expression of query.

Relational algebra has operators that describes queries and DBMS uses these expressions in query optimizing (Figure 41).

When any query comes to SQL Server to execute, some processes are passed to get result. These steps are;

- First SQL Server parse the query that means it checks syntax of query.
- Than after validation of query (assuming the query is valid) that means verifies objects referred to are database objects and requested operations are valid, it translate query into a relational algebra to use it in query optimization process.
- Relational algebra in SQL Server likes logical query tree and the tree occurs with logical operators of relational algebra.
- Recompose relational algebra operations to find most useful query plan.
- The next step is to choose a plan that is less costly, estimate the cost of query with using relational algebra operations and the Query Optimizer creates the optimal execution plan for the query.

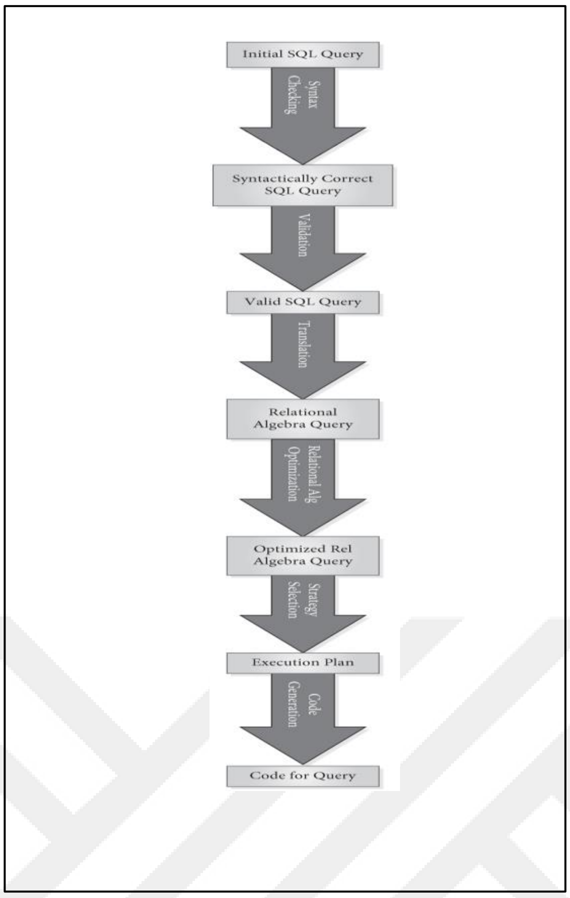


Figure 41. Place of Relational Algebra Query in the Query Optimization (Ricardo, Urban, 2016).

Therefore, Relational Algebra is important to understand and optimize the query in relational database. Also, designing the tables in the database correctly is important in terms of query performance. To understand relational algebra simply will be created a new simple relational database and will be some query executions with relational algebra operators and then all query trees and algebraically expressions of queries (Query one, two, three, four and five) that used previous chapter will be shown.

The example database name is Dental Clinic. Class diagram and ER diagram of database are shown in Figure 42 and Figure 43:

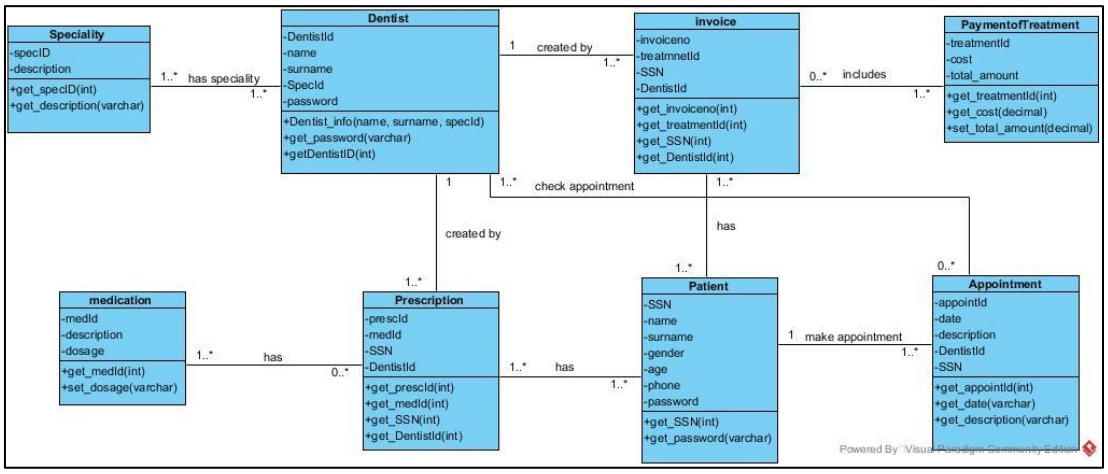


Figure 42. Class Diagram of Dental Clinic Database.

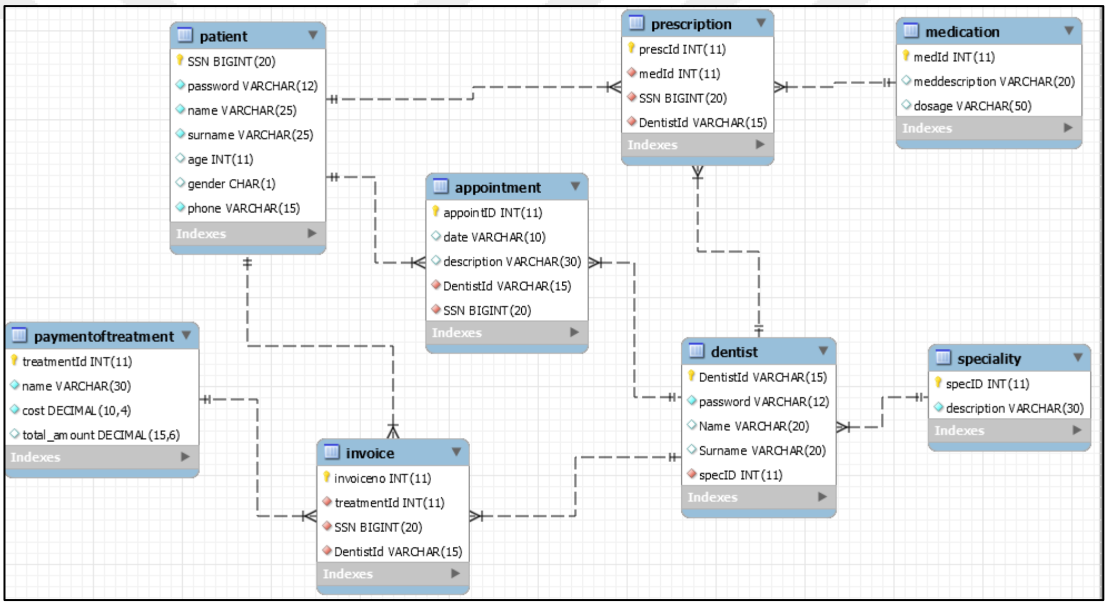


Figure 43. ER Diagram of Dental Clinic Database.

Additionally, a diagram can be created with using MS SQL Server (Figure 44).

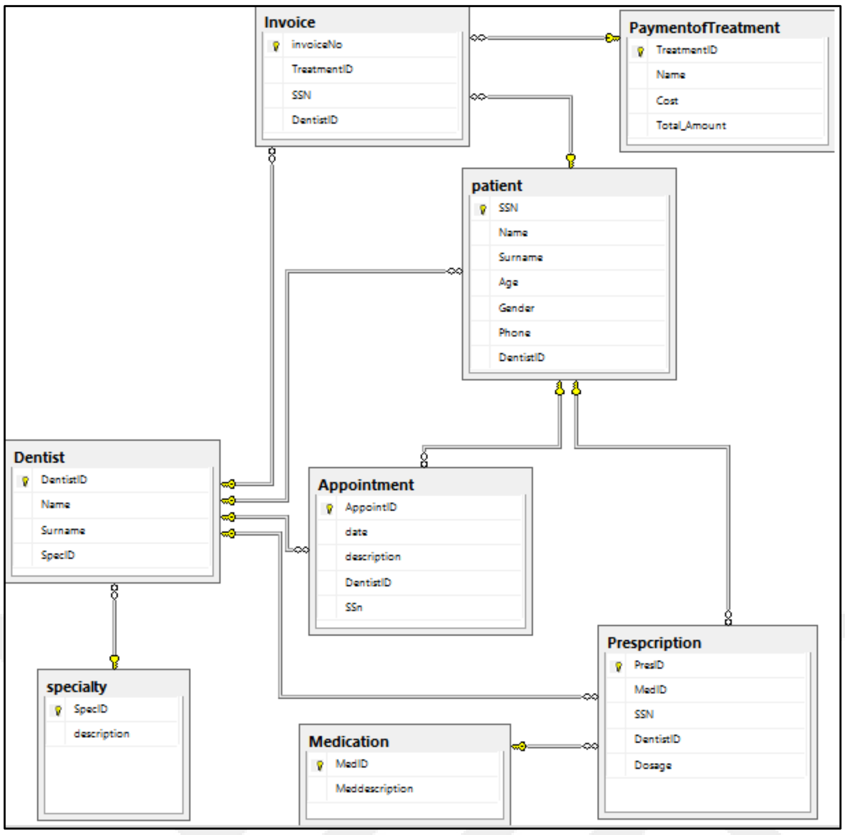


Figure 44. Dental Clinic Database Diagram from MS SQL Server.

As seen in Figure 44, database has dentist, patient, specialty, prescription, appointment, paymentoftreatment, medication, and invoice tables. To understand relational algebra operators these tables will be used. Before do some example the operations of relational algebra can be defined.

3.1 RELATIONAL ALGEBRA MAIN OPERATORS

a. SELECT Operation σ:

It fetches all the information (subsets) on the table or just the desired part from relation. This operation result can be the input for another relational algebra operation (Ramakrishnan, Gehrke).

b. PROJECT Operation Π :

Projection operation deletes attributes that are not in projection list. Duplicate rows are removed from result because relations are sets. So, it fetches certain columns on the selection operation.

c. CARTESIAN PRODUCT Operation ⋈ :

Each row of table 1 is paired each row of table 2. Result schema has all information of relational tables of table 1 and table 2.

d. UNION Operation ∪ :

Returns all the tuples from two different relational tables. The two relations must be union compatible.

e. DIFERENCE Operations — :

For two relational tables to be applicable tables must have the same arity and Attribute domains must be compatible.

f. RENAME Operation ρ :

Returns an identical result to R except that in all tuples attribute

A is renamed to B (McBrien, 2010). It used to rename the table after Cartesian product operation or join operations.

g. INTERSECTION Operation ∩ :

Fetches common tuples between two relational tables. Tables must be compatible.

h. JOIN Operation ⋈ :

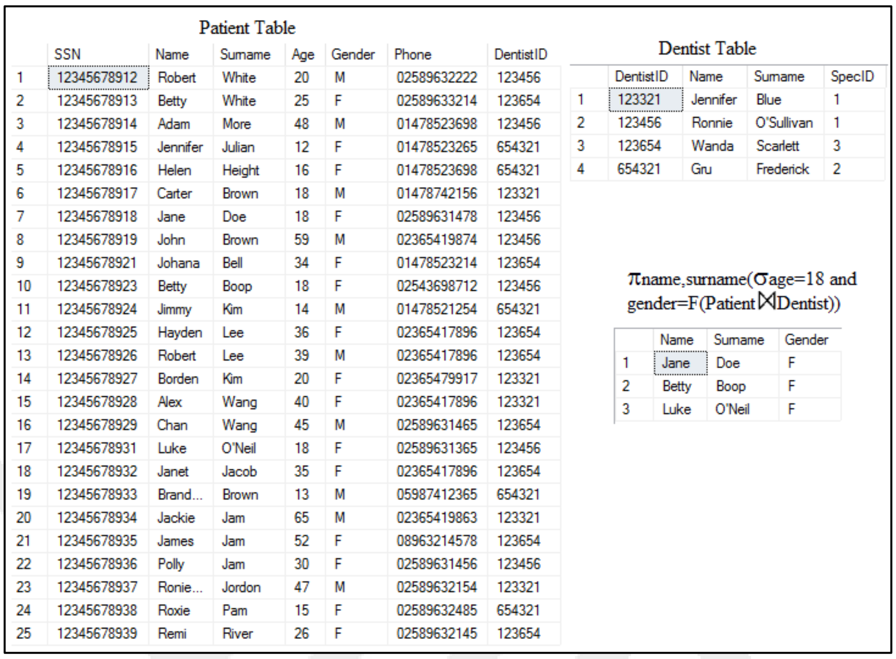
Returns tuples from the Cartesian product where common attributes between relational tables have the same values and removes duplicated columns (McBien, 2010). When more than one table information is available, a single table can be presented it like one table.

All SQL queries can be shown as relational expressions. It can be explained with some examples:

- List all 18 years old and female patients Name and Surname which are patient of Dr.O'Sullivan. Figure 45 represents the result of query.
  SQL:
  Select p.Name, p.Surname
  From Patient p join Dentist d on p.DentistID=d.DentistID
  Where p.Age=18 and p.Gender='F'

Relational Algebra:

Πname,surname(σage=18 and gender=F(Patient ⋈ Dentist))



Patient Table

	SSN	Name	Surname	Age	Gender	Phone	DentistID
1	12345678912	Robert	White	20	M	02589632222	123456
2	12345678913	Betty	White	25	F	02589633214	123654
3	12345678914	Adam	More	48	M	01478523698	123456
4	12345678915	Jennifer	Julian	12	F	01478523265	654321
5	12345678916	Helen	Height	16	F	01478523698	654321
6	12345678917	Carter	Brown	18	M	01478742156	123321
7	12345678918	Jane	Doe	18	F	02589631478	123456
8	12345678919	John	Brown	59	M	02365419874	123456
9	12345678921	Johana	Bell	34	F	01478523214	123654
10	12345678923	Betty	Boop	18	F	02543698712	123456
11	12345678924	Jimmy	Kim	14	M	01478521254	654321
12	12345678925	Hayden	Lee	36	F	02365417896	123654
13	12345678926	Robert	Lee	39	M	02365417896	123654
14	12345678927	Borden	Kim	20	F	02365479917	123321
15	12345678928	Alex	Wang	40	F	02365417896	123321
16	12345678929	Chan	Wang	45	M	02589631465	123654
17	12345678931	Luke	O'Neil	18	F	02589631365	123456
18	12345678932	Janet	Jacob	35	F	02365417896	123654
19	12345678933	Brand...	Brown	13	M	05987412365	654321
20	12345678934	Jackie	Jam	65	M	02365419863	123321
21	12345678935	James	Jam	52	F	08963214578	123654
22	12345678936	Polly	Jam	30	F	02589631456	123456
23	12345678937	Ronie...	Jordon	47	M	02589632154	123321
24	12345678938	Roxie	Pam	15	F	02589632485	654321
25	12345678939	Remi	River	26	F	02589632145	123654

Dentist Table

	DentistID	Name	Surname	SpecID
1	123321	Jennifer	Blue	1
2	123456	Ronnie	O'Sullivan	1
3	123654	Wanda	Scarlett	3
4	654321	Gru	Frederick	2

πname,surname(σage=18 and gender=F(Patient ⋈ Dentist))

	Name	Surname	Gender
1	Jane	Doe	F
2	Betty	Boop	F
3	Luke	O'Neil	F

Figure 45. Representation of Query Result.

- List all patients whom are have same Surname and their ages, their doctors' name and specialties. Figure 46 represents the result of query.

  SQL:

  Select p.Name,p.age,d.Name, s.description

  From patient p inner join Dentist d on p.DentistID=d.DentistID inner join Specialty s on s.SpecID=d.SpecID

  Where p.surname in (SELECT Surname

  FROM patient

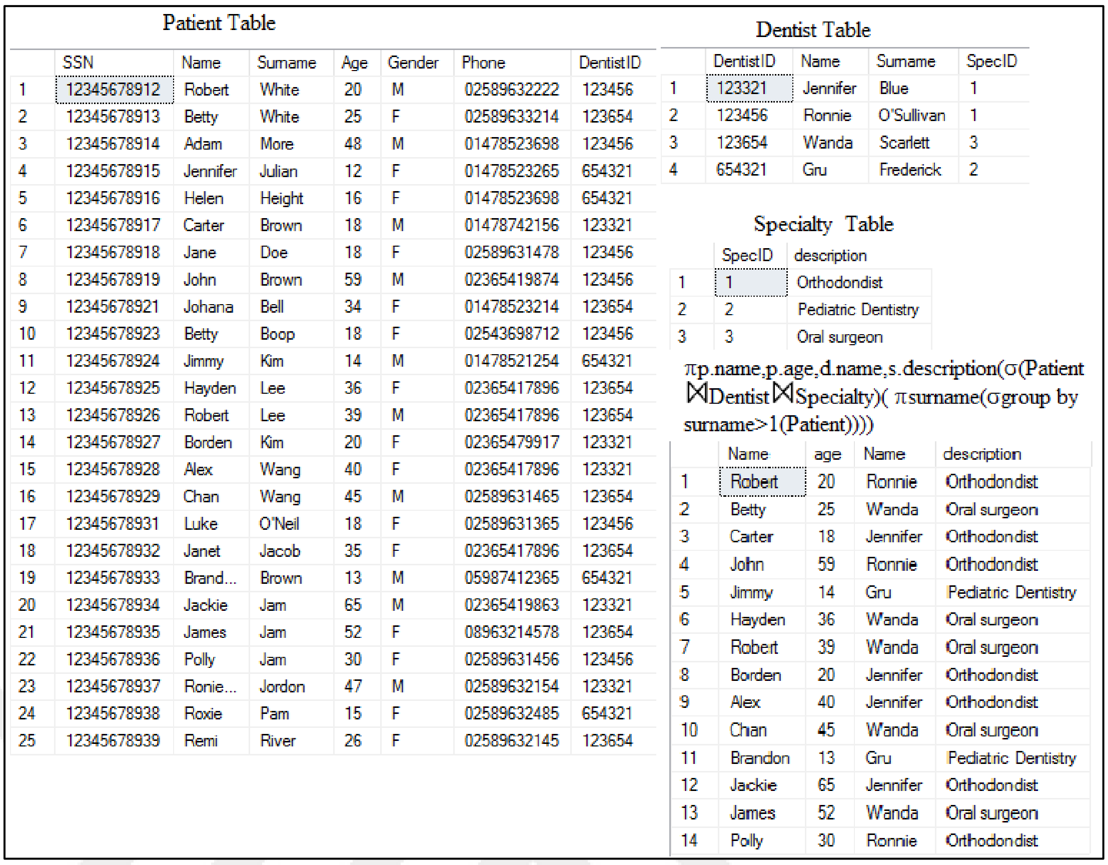
  GROUP BY Surname

  HAVING

  COUNT(Surname) > 1)

  Relational Algebra:

  Πp.name,p.age,d.name,s.description(σ (Patient ⋈ Dentist ⋈ Specialty)(Πsurname(σgroup by surname>1(Patient))))


Patient Table

	SSN	Name	Surname	Age	Gender	Phone	DentistID
1	12345678912	Robert	White	20	M	02589632222	123456
2	12345678913	Betty	White	25	F	02589633214	123654
3	12345678914	Adam	More	48	M	01478523698	123456
4	12345678915	Jennifer	Julian	12	F	01478523265	654321
5	12345678916	Helen	Height	16	F	01478523698	654321
6	12345678917	Carter	Brown	18	M	01478742156	123321
7	12345678918	Jane	Doe	18	F	02589631478	123456
8	12345678919	John	Brown	59	M	02365419874	123456
9	12345678921	Johana	Bell	34	F	01478523214	123654
10	12345678923	Betty	Boop	18	F	02543698712	123456
11	12345678924	Jimmy	Kim	14	M	01478521254	654321
12	12345678925	Hayden	Lee	36	F	02365417896	123654
13	12345678926	Robert	Lee	39	M	02365417896	123654
14	12345678927	Borden	Kim	20	F	02365479917	123321
15	12345678928	Alex	Wang	40	F	02365417896	123321
16	12345678929	Chan	Wang	45	M	02589631465	123654
17	12345678931	Luke	O'Neil	18	F	02589631365	123456
18	12345678932	Janet	Jacob	35	F	02365417896	123654
19	12345678933	Brand...	Brown	13	M	05987412365	654321
20	12345678934	Jackie	Jam	65	M	02365419863	123321
21	12345678935	James	Jam	52	F	08963214578	123654
22	12345678936	Polly	Jam	30	F	02589631456	123456
23	12345678937	Ronie...	Jordon	47	M	02589632154	123321
24	12345678938	Roxie	Pam	15	F	02589632485	654321
25	12345678939	Remi	River	26	F	02589632145	123654

Dentist Table

	DentistID	Name	Surname	SpecID
1	123321	Jennifer	Blue	1
2	123456	Ronnie	O'Sullivan	1
3	123654	Wanda	Scarlett	3
4	654321	Gru	Frederick	2

Specialty Table

	SpecID	description
1	1	Orthodondist
2	2	Pediatric Dentistry
3	3	Oral surgeon

πp.name,p.age,d.name,s.description(σ(Patient ⋈Dentist ⋈Specialty)(πsurname(σgroup by surname>1(Patient))))

	Name	age	Name	description
1	Robert	20	Ronnie	Orthodondist
2	Betty	25	Wanda	Oral surgeon
3	Carter	18	Jennifer	Orthodondist
4	John	59	Ronnie	Orthodondist
5	Jimmy	14	Gru	Pediatric Dentistry
6	Hayden	36	Wanda	Oral surgeon
7	Robert	39	Wanda	Oral surgeon
8	Borden	20	Jennifer	Orthodondist
9	Alex	40	Jennifer	Orthodondist
10	Chan	45	Wanda	Oral surgeon
11	Brandon	13	Gru	Pediatric Dentistry
12	Jackie	65	Jennifer	Orthodondist
13	James	52	Wanda	Oral surgeon
14	Polly	30	Ronnie	Orthodondist

Figure 46. Representation of Query Result.

- List all Orthodontics patient and their treatment. Figure 47 represents the result of query.

  SQL:

  Select p.Name,p.Surname,p.Age,p.Gender,a.description

  From patient p join Appointment a on p.SSN=a.SSn

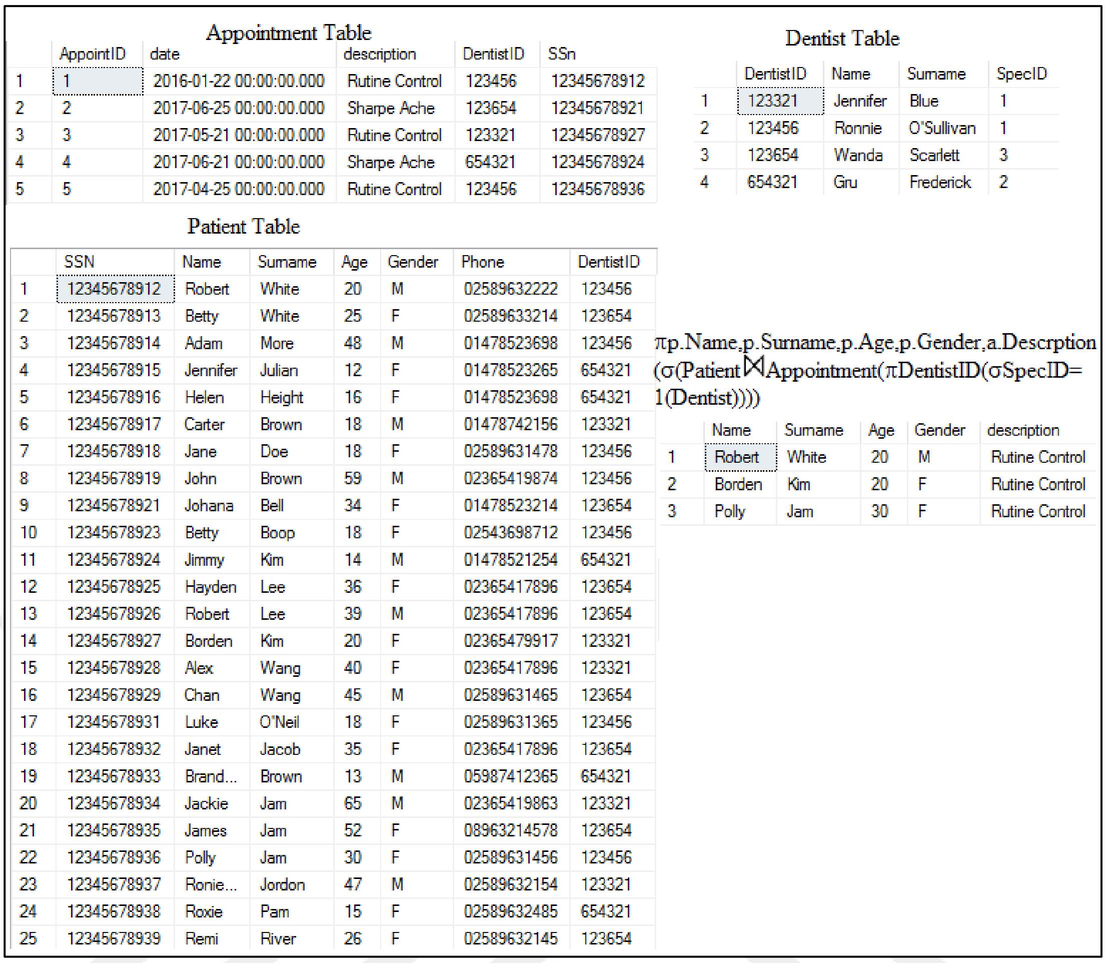
  Where a.DentistID in (Select DentistID

  From Dentist

  Where SpecID='1')

  Relational Algebra:

  Πp.Name,p.Surname,p.Age,p.Gender,a.Description(σ(Patient ⋈ Appointment)(ΠDentistID(σSpecID=1(Dentist)))



Appointment Table

	AppointID	date	description	DentistID	SSn
1	1	2016-01-22 00:00:00.000	Rutine Control	123456	12345678912
2	2	2017-06-25 00:00:00.000	Sharpe Ache	123654	12345678921
3	3	2017-05-21 00:00:00.000	Rutine Control	123321	12345678927
4	4	2017-06-21 00:00:00.000	Sharpe Ache	654321	12345678924
5	5	2017-04-25 00:00:00.000	Rutine Control	123456	12345678936

Dentist Table

	DentistID	Name	Surname	SpecID
1	123321	Jennifer	Blue	1
2	123456	Ronnie	O'Sullivan	1
3	123654	Wanda	Scarlett	3
4	654321	Gru	Frederick	2

Patient Table

	SSN	Name	Surname	Age	Gender	Phone	DentistID
1	12345678912	Robert	White	20	M	02589632222	123456
2	12345678913	Betty	White	25	F	02589633214	123654
3	12345678914	Adam	More	48	M	01478523698	123456
4	12345678915	Jennifer	Julian	12	F	01478523265	654321
5	12345678916	Helen	Height	16	F	01478523698	654321
6	12345678917	Carter	Brown	18	M	01478742156	123321
7	12345678918	Jane	Doe	18	F	02589631478	123456
8	12345678919	John	Brown	59	M	02365419874	123456
9	12345678921	Johana	Bell	34	F	01478523214	123654
10	12345678923	Betty	Boop	18	F	02543698712	123456
11	12345678924	Jimmy	Kim	14	M	01478521254	654321
12	12345678925	Hayden	Lee	36	F	02365417896	123654
13	12345678926	Robert	Lee	39	M	02365417896	123654
14	12345678927	Borden	Kim	20	F	02365479917	123321
15	12345678928	Alex	Wang	40	F	02365417896	123321
16	12345678929	Chan	Wang	45	M	02589631465	123654
17	12345678931	Luke	O'Neil	18	F	02589631365	123456
18	12345678932	Janet	Jacob	35	F	02365417896	123654
19	12345678933	Brand...	Brown	13	M	05987412365	654321
20	12345678934	Jackie	Jam	65	M	02365419863	123321
21	12345678935	James	Jam	52	F	08963214578	123654
22	12345678936	Polly	Jam	30	F	02589631456	123456
23	12345678937	Ronie...	Jordon	47	M	02589632154	123321
24	12345678938	Roxie	Pam	15	F	02589632485	654321
25	12345678939	Remi	River	26	F	02589632145	123654

πp.Name,p.Surname,p.Age,p.Gender,a.Descrption (σ(Patient ⋈ Appointment(πDentistID(σSpecID= 1(Dentist)))

	Name	Surname	Age	Gender	description
1	Robert	White	20	M	Rutine Control
2	Borden	Kim	20	F	Rutine Control
3	Polly	Jam	30	F	Rutine Control

Figure 47. Representation of Query Result.

- List Dr. Gru Frederick's patients' name, gender and ages. Figure 48 represents the result of query.

  <u>SQL:</u>

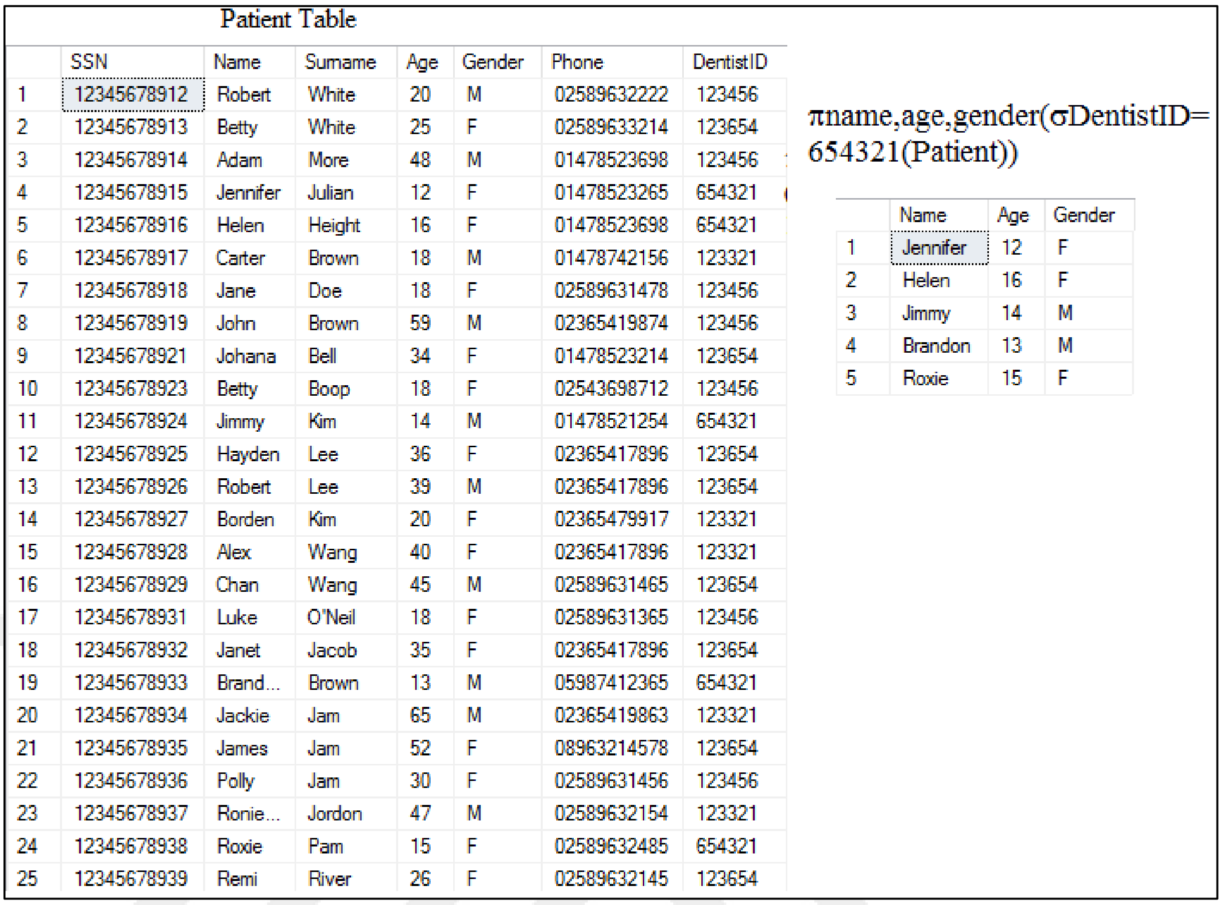
  Select Name,Age,Gender

  From patient

  Where DentistID = '654321'

  <u>Relational Algebra:</u>

  Πname,age,gender(σDentistID=654321(Patient))


Patient Table

	SSN	Name	Surname	Age	Gender	Phone	DentistID
1	12345678912	Robert	White	20	M	02589632222	123456
2	12345678913	Betty	White	25	F	02589633214	123654
3	12345678914	Adam	More	48	M	01478523698	123456
4	12345678915	Jennifer	Julian	12	F	01478523265	654321
5	12345678916	Helen	Height	16	F	01478523698	654321
6	12345678917	Carter	Brown	18	M	01478742156	123321
7	12345678918	Jane	Doe	18	F	02589631478	123456
8	12345678919	John	Brown	59	M	02365419874	123456
9	12345678921	Johana	Bell	34	F	01478523214	123654
10	12345678923	Betty	Boop	18	F	02543698712	123456
11	12345678924	Jimmy	Kim	14	M	01478521254	654321
12	12345678925	Hayden	Lee	36	F	02365417896	123654
13	12345678926	Robert	Lee	39	M	02365417896	123654
14	12345678927	Borden	Kim	20	F	02365479917	123321
15	12345678928	Alex	Wang	40	F	02365417896	123321
16	12345678929	Chan	Wang	45	M	02589631465	123654
17	12345678931	Luke	O'Neil	18	F	02589631365	123456
18	12345678932	Janet	Jacob	35	F	02365417896	123654
19	12345678933	Brand...	Brown	13	M	05987412365	654321
20	12345678934	Jackie	Jam	65	M	02365419863	123321
21	12345678935	James	Jam	52	F	08963214578	123654
22	12345678936	Polly	Jam	30	F	02589631456	123456
23	12345678937	Ronie...	Jordon	47	M	02589632154	123321
24	12345678938	Roxie	Pam	15	F	02589632485	654321
25	12345678939	Remi	River	26	F	02589632145	123654

πname,age,gender(σDentistID=654321(Patient))

	Name	Age	Gender
1	Jennifer	12	F
2	Helen	16	F
3	Jimmy	14	M
4	Brandon	13	M
5	Roxie	15	F

Figure 48. Representation of Query Result.

The simple database and simple queries are created to understand how operation works. Talked about relational algebra place in query optimization. Query tree of queries of Dental Clinic Database (Figure 49, Figure 50, Figure 51 and Figure 52) will be created to show how tree will be created. The reason for the creation of the query tree is to make it easier to understand how the relational algebra is constructed. Query Tree occurs with leaf nodes that are relations. Reading the expressions begin bottom and ends root node (top).

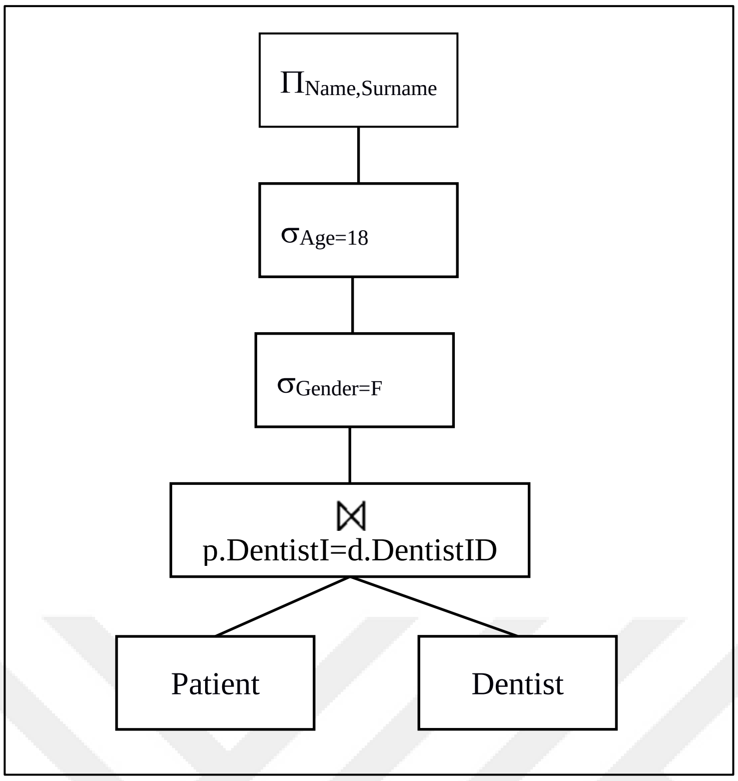


Figure 49. Query Tree of name,surname(σage=18 and gender=F(Patient⋈Dentist))

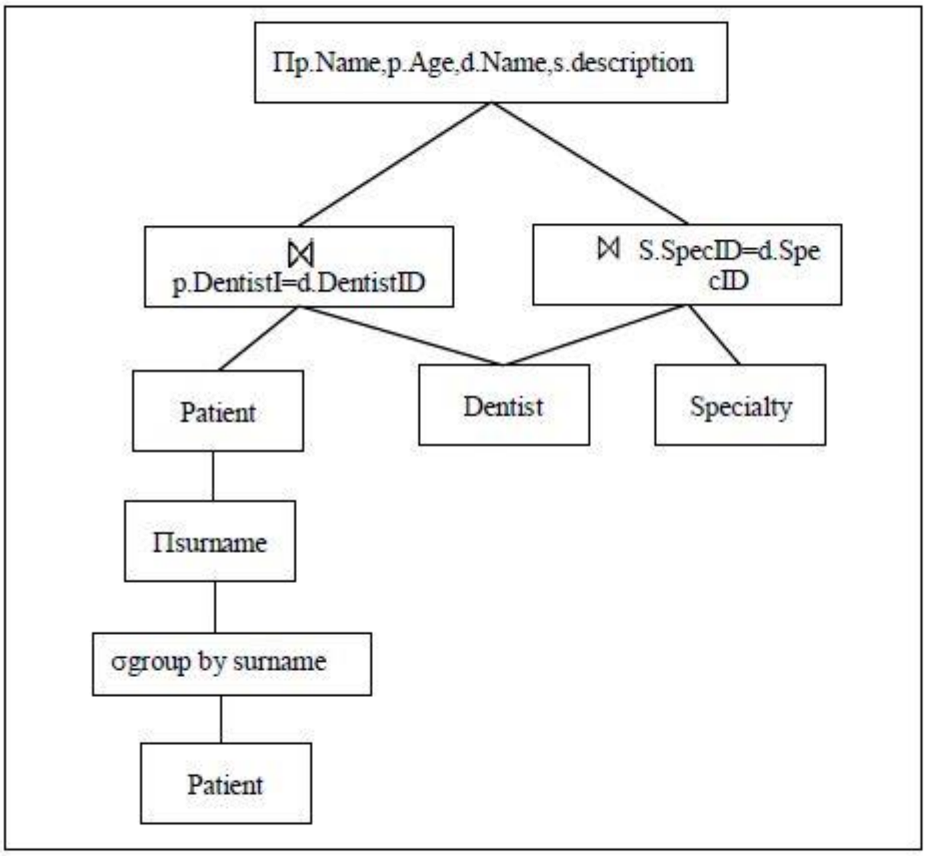


Figure 50. Query Tree of Πp.name,p.age,d.name,s.description(σ (Patient⋈Dentist ⋈Specialty)(Πsurname(σgroup by surname>1(Patient))))

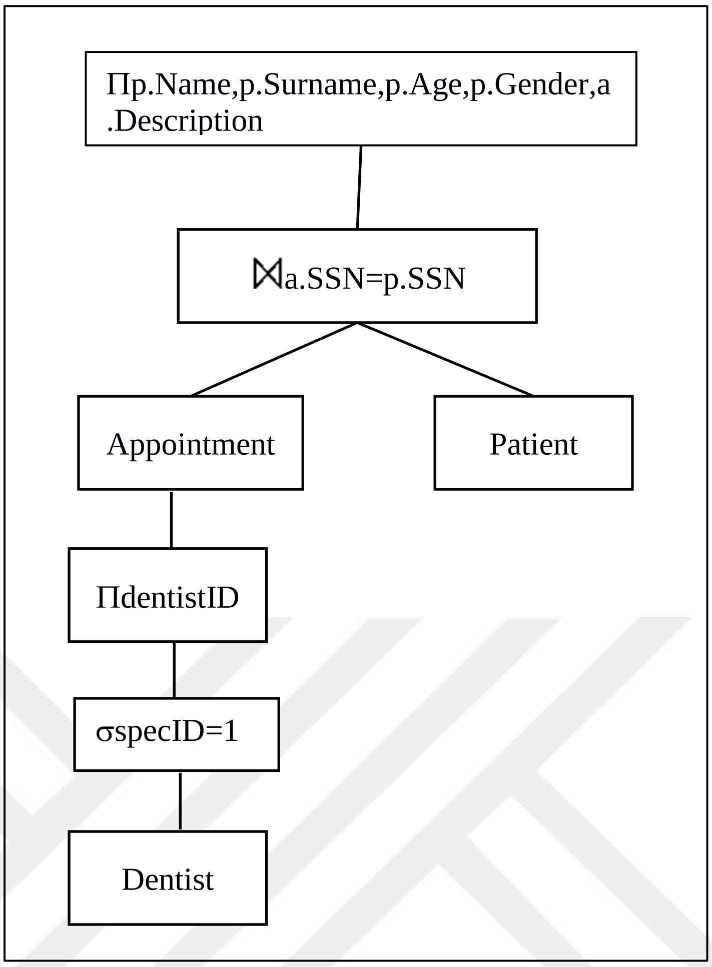


Figure 51. Query Tree of Πp.Name,p.Surname,p.Age,p.Gender,a.Description(σ (Patient⋈Appointment)(ΠDentistID(σSpecID=1(Dentist))))

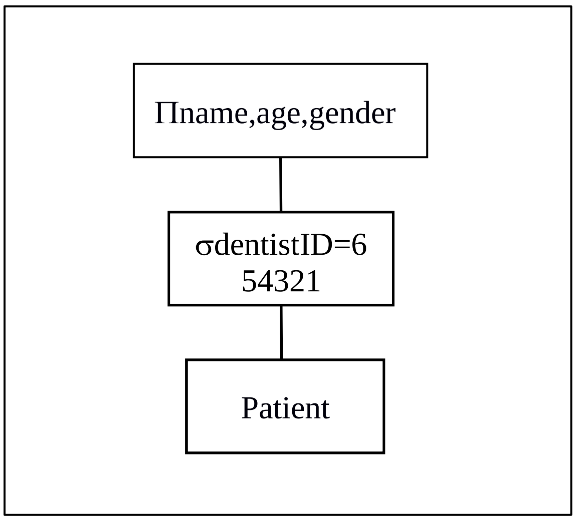


Figure 52. Query Tree of Πname, age, gender(σDentistID=654321(Patient))

3.2 QUERIES' RELATIONAL ALGEBRA OPERATIONS AND QUERY TREE REPRESENTATIONS

These study's queries will be examined as relational algebra operations and their query tree representations.

1. SQL:

   Select *

   From person.Person

   Relational Algebra:

   σ (Person.person)

2. SQL:

   Select *

   From sales.Customer

   Relational Algebra:

   σ (sales.customer)

3. SQL:

   Select StoreID

   From Sales.Customer

   Where StoreID Is Not Null and AccountNumber like '%17'

   Relational Algebra:

   ΠStoreID(σStoreID Is Not Null AND AccountNumber like '%17'(sales.customer))

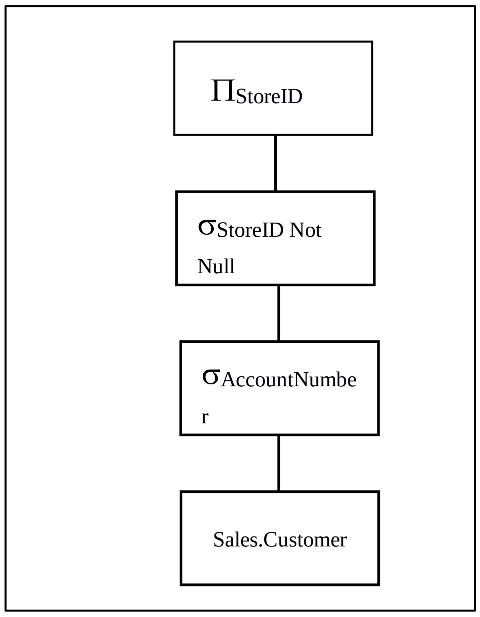


Figure 53. Query Tree of Query Three.

4. SQL:

   Select PersonID,Title,FirstName,LastName

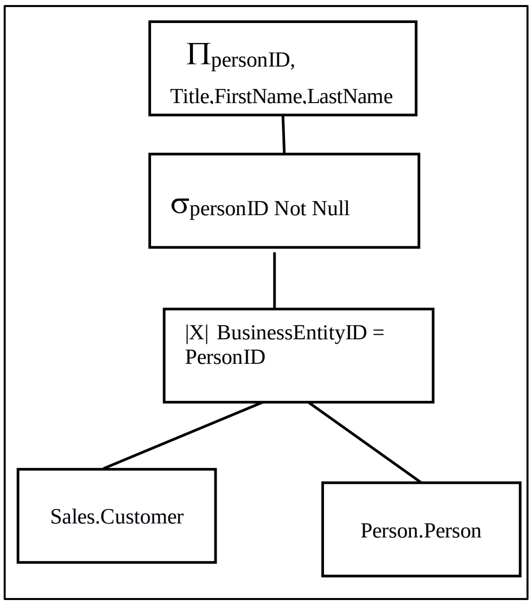
   From Sales.Customer As sc Inner Join

   Person.Person As pp On pp.BusinessEntityID = sc.PersonID

   Where PersonID Is Not Null

   Relational Algebra:

   ΠpersonID,Title,FirstName,LastName(σpersonID Is Not Null (sales.customer |X| Person.Person))



$\Pi_{personID, Title,FirstName,LastName}$
$\sigma_{personID\ Not\ Null}$
|X| BusinessEntityID = PersonID
Sales.Customer
Person.Person

Figure 54. Query Tree of Query Four.

5. SQL:

   Select StoreID, FirstName,MiddleName,LastName

   From Sales.Customer as sc join Person.Person as pp on

   sc.PersonID=pp.BusinessEntityID

   Where StoreID Is Not Null and AccountNumber like '%17' and

   StoreID<1000

   Relational Algebra:

ΠStoreID,FirstName,MiddleName,LastName(σStoreID Is Not Null AND AccountNumber like '%17' AND StoreID < 1000(sales.customer |X| Person.Person))

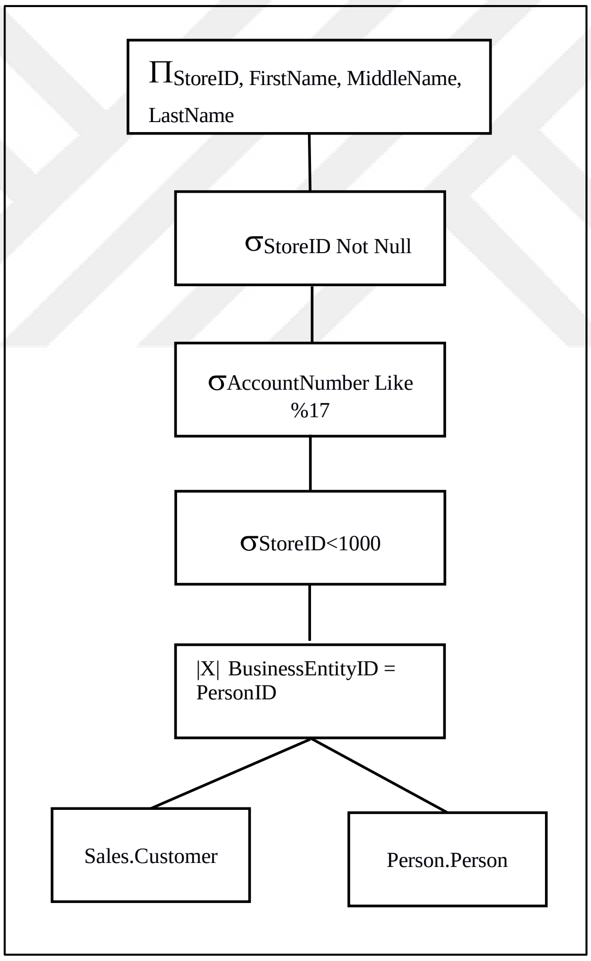


Figure 55. Query Tree of Query Five.

CHAPTER FOUR
USING DYNAMIC MANAGEMENT VIEW (DMV) AND DYNAMIC MANAGEMENT FUNCTIONS (DMF) FOR OPTIMIZATION

Dynamic management views and functions return server state information that can be used to monitor the health of a server instance, diagnose problems, and tune performance. With using DMV and DMFs, users can scan all necessary information to manage the database. DMV has querying structure, DMF returns table with using parameter. Users can find useful information with using DMV and DMF like missing index, most expensive index, least used index, most expensive queries, and most used queries. DMV has spell dictionary, all DMVs are drawn from system table and this system table shows on our DMV as a sys.dm_ and after this notation db_, exec_, io_, os_, are followed. They refer groups of what users looking for. Mostly sys.dm_db_ and sys.dm_exec_ DMVs will be used in this study.

sys.dm_db_: It quotes chapter and verse about database and index.

sys.dm_exec_: They inform directly or indirectly about the relevant elements by the execution of user codes (Acungil, 2016).

There will be shown and explained some useful DMV and DMFs to optimize queries. Users can find slow queries, running queries, missing index and execution plan of queries, fragmentation of index, most CPU consuming queries and most I/O performed queries. After finding and fixing all of these problems, the query performance will be probably improved.

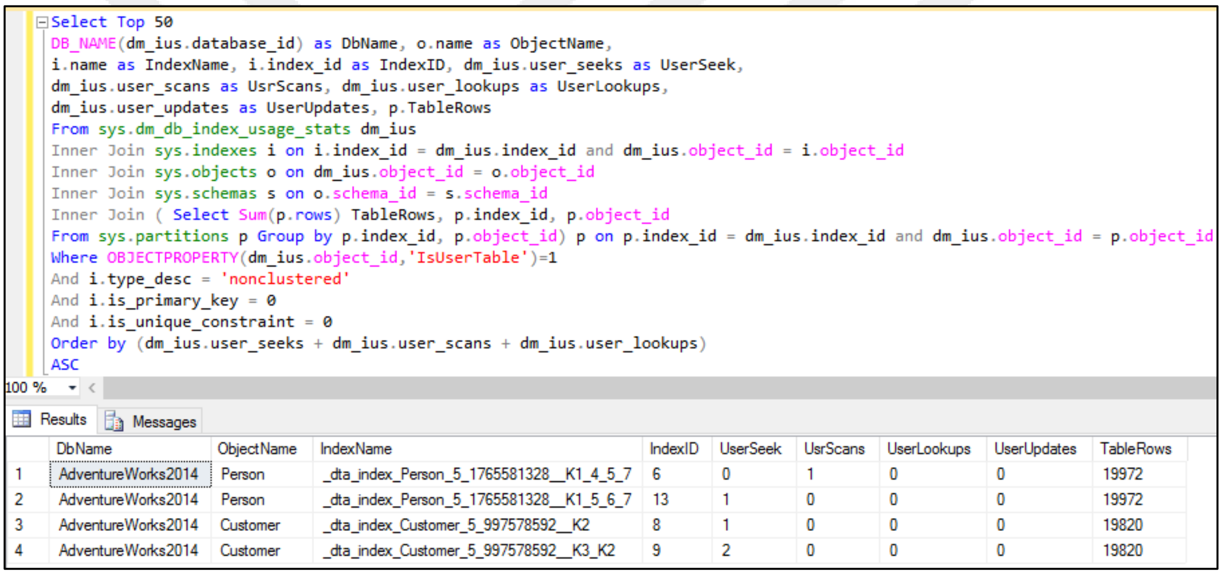
As stated previously, index and their importance for query performance previous chapter, let’s remember how index were used; profiler can be used to trace query or activity monitor can be used to find expensive queries and users can directly use database engine tuning administer on query to get recommendation and apply directly or examine recommended indexes. When users do some DML operation on queries it effects index and add or drop operations may effect query performance so users need to control index, which are unused, missing, or there is any fragmentation on index.

So, to control these query problems DMV and DMF querying can be used.

4.1 UNUSED INDEX

Finding unused index increases the performance because all deletion and addition operations also effects index. For this reason, these indexes should be controlled periodically. To find unused or rarely used index users can run **sys.dm_db_index_usage_stats** DMV.

With this query, users can get more information about our database’s indexes situations. After running this query, users will get some information columns and rows about user seeks, user scans and user updates (Figure 56).



```sql
Select Top 50
DB_NAME(dm_ius.database_id) as DbName, o.name as ObjectName,
i.name as IndexName, i.index_id as IndexID, dm_ius.user_seeks as UserSeek,
dm_ius.user_scans as UsrScans, dm_ius.user_lookups as UserLookups,
dm_ius.user_updates as UserUpdates, p.TableRows
From sys.dm_db_index_usage_stats dm_ius
Inner Join sys.indexes i on i.index_id = dm_ius.index_id and dm_ius.object_id = i.object_id
Inner Join sys.objects o on dm_ius.object_id = o.object_id
Inner Join sys.schemas s on o.schema_id = s.schema_id
Inner Join ( Select Sum(p.rows) TableRows, p.index_id, p.object_id
From sys.partitions p Group by p.index_id, p.object_id) p on p.index_id = dm_ius.index_id and dm_ius.object_id = p.object_id
Where OBJECTPROPERTY(dm_ius.object_id,'IsUserTable')=1
And i.type_desc = 'nonclustered'
And i.is_primary_key = 0
And i.is_unique_constraint = 0
Order by (dm_ius.user_seeks + dm_ius.user_scans + dm_ius.user_lookups)
ASC
```

100 %

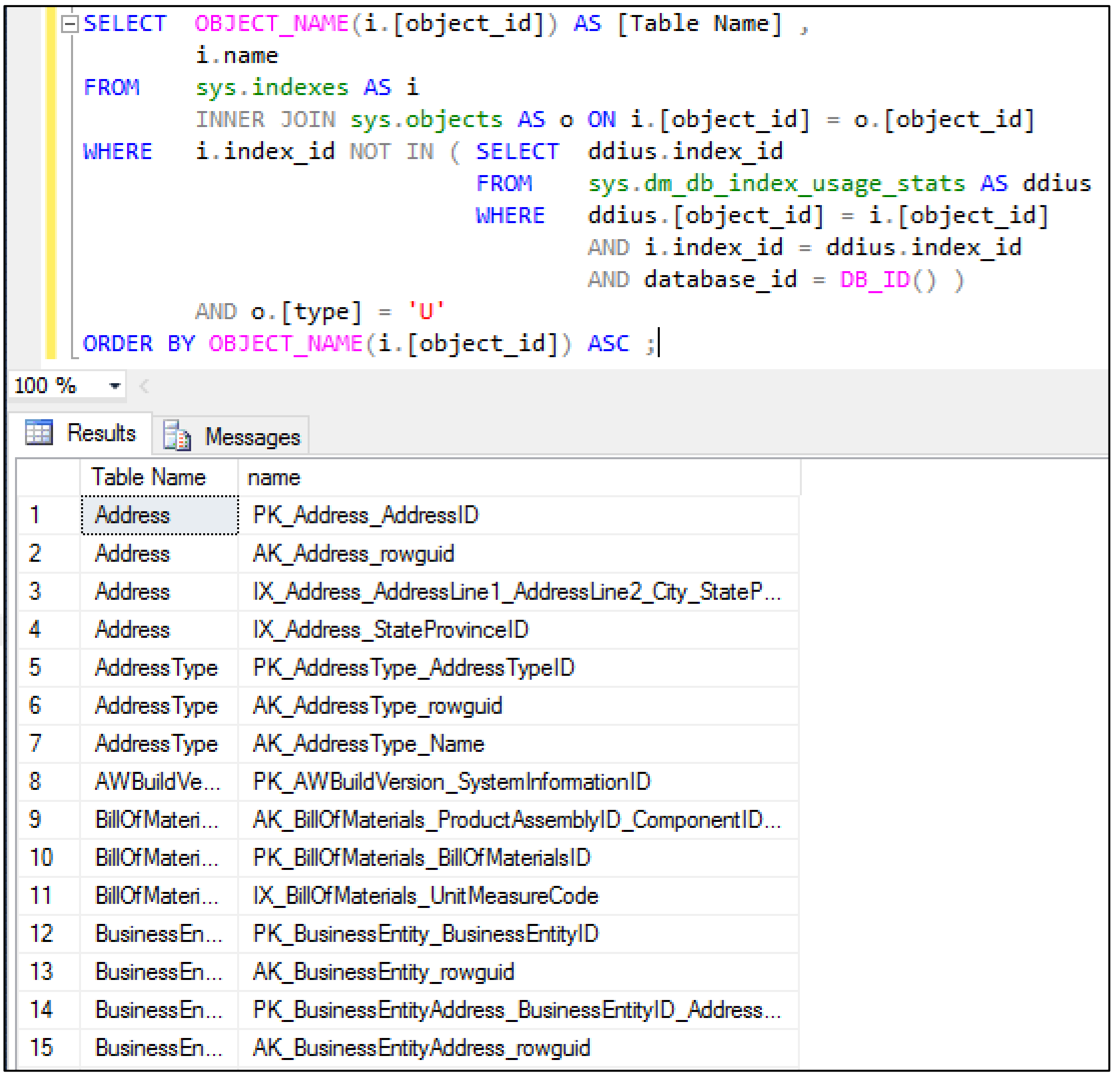
Results | Messages

	DbName	ObjectName	IndexName	IndexID	UserSeek	UsrScans	UserLookups	UserUpdates	TableRows
1	AdventureWorks2014	Person	_dta_index_Person_5_1765581328__K1_4_5_7	6	0	1	0	0	19972
2	AdventureWorks2014	Person	_dta_index_Person_5_1765581328__K1_5_6_7	13	1	0	0	0	19972
3	AdventureWorks2014	Customer	_dta_index_Customer_5_997578592__K2	8	1	0	0	0	19820
4	AdventureWorks2014	Customer	_dta_index_Customer_5_997578592__K3_K2	9	2	0	0	0	19820

Figure 56. Querying of Unused Index.

With using this output of query, users can control last index usage information. However, most important one is UserUpdates, if column has very high value and index did not use, users must do some operation on this index, because this situation can cause of low performance.

It could get similar result with using different DMV querying (Figure 57). With using this querying, users can get name of unused index. Using these information users can eliminate some of unused indexes.



```
SELECT  OBJECT_NAME(i.[object_id]) AS [Table Name] ,
        i.name
FROM    sys.indexes AS i
        INNER JOIN sys.objects AS o ON i.[object_id] = o.[object_id]
WHERE   i.index_id NOT IN ( SELECT  ddius.index_id
                            FROM    sys.dm_db_index_usage_stats AS ddius
                            WHERE   ddius.[object_id] = i.[object_id]
                                    AND i.index_id = ddius.index_id
                                    AND database_id = DB_ID() )
        AND o.[type] = 'U'
ORDER BY OBJECT_NAME(i.[object_id]) ASC ;
```

	Table Name	name
1	Address	PK_Address_AddressID
2	Address	AK_Address_rowguid
3	Address	IX_Address_AddressLine1_AddressLine2_City_StateP...
4	Address	IX_Address_StateProvinceID
5	AddressType	PK_AddressType_AddressTypeID
6	AddressType	AK_AddressType_rowguid
7	AddressType	AK_AddressType_Name
8	AWBuildVe...	PK_AWBuildVersion_SystemInformationID
9	BillOfMateri...	AK_BillOfMaterials_ProductAssemblyID_ComponentID...
10	BillOfMateri...	PK_BillOfMaterials_BillOfMaterialsID
11	BillOfMateri...	IX_BillOfMaterials_UnitMeasureCode
12	BusinessEn...	PK_BusinessEntity_BusinessEntityID
13	BusinessEn...	AK_BusinessEntity_rowguid
14	BusinessEn...	PK_BusinessEntityAddress_BusinessEntityID_Address...
15	BusinessEn...	AK_BusinessEntityAddress_rowguid

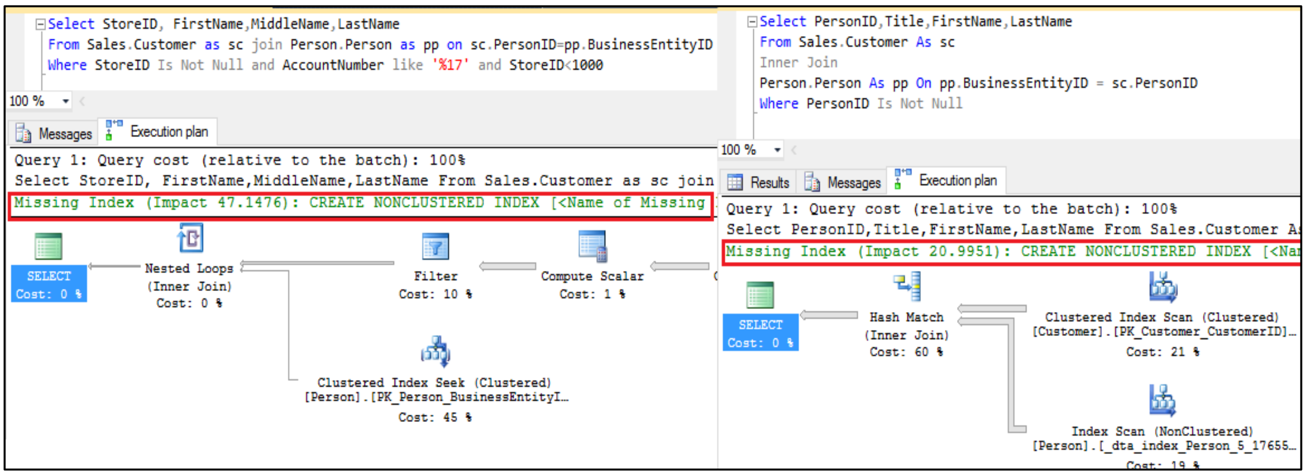
Figure 57. Querying of Unused index.

4.2 MISSING INDEX

Creating index is very important to database and its performance because it can be said that index is a shortcut of reaching query results. If index is created correctly, it will be work fast. Previously talked about query execution previous chapter when users run the query, optimizer will try to find short and fast way to bring the result and it does this with indexes. So, when optimizer try to find short and fast way to query and there is no index to use it will identify index to speed up to performance, and users see this recommended index as a missing index on our screen. It can be also seen this warning on query execution plan graph (Figure 58). Finding and applying missing index can increase the query performance of course it can be checked before apply permanently with wrong usage it will effect negatively the performance.

DMV and DMF can be used to identify missing indexes. There are three DMV, one DMF.

- **sys.dm_db_missing_index_details:** This DMV shows which chosen index will be use on which table and column by optimizer.
- **sys.dm_db_missing_index_group_stats:** This DMV shows summary information of groups of missing index.
- **sys.dm_db_missing_index_groups:** This DMV is a kind of bridge between sys.dm_db_missing_index_details and sys.dm_db_missing_index_group_stats. These two DMVs create a meaningful whole with this DMV.
- **sys.dm_db_missing_index_columns:** a DMF that accepts an index_handle parameter and returns a table providing details of columns that would comprise the
  suggested missing index (Davidson & Ford, 2010). It shows information about database table columns that are missing index, excluding spatial indexes.



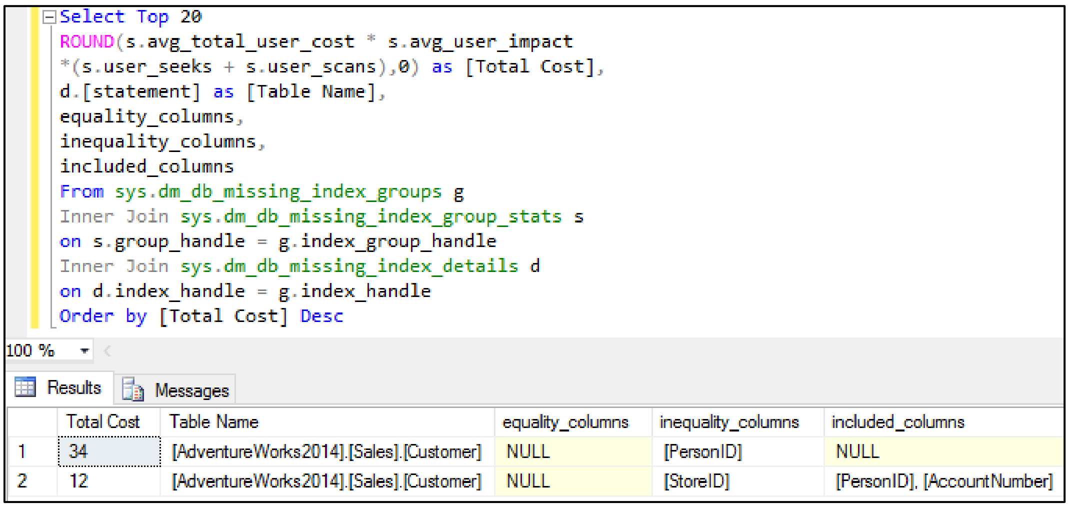
```
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
```

```
Query 1: Query cost (relative to the batch): 100%
Select StoreID, FirstName,MiddleName,LastName From Sales.Customer as sc join
Missing Index (Impact 47.1476): CREATE NONCLUSTERED INDEX [<Name of Missing
```

```
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
```

```
Query 1: Query cost (relative to the batch): 100%
Select PersonID,Title,FirstName,LastName From Sales.Customer A
Missing Index (Impact 20.9951): CREATE NONCLUSTERED INDEX [<Na
```

Figure 58. Missing Index Warnings on Query Execution Plan Graph.



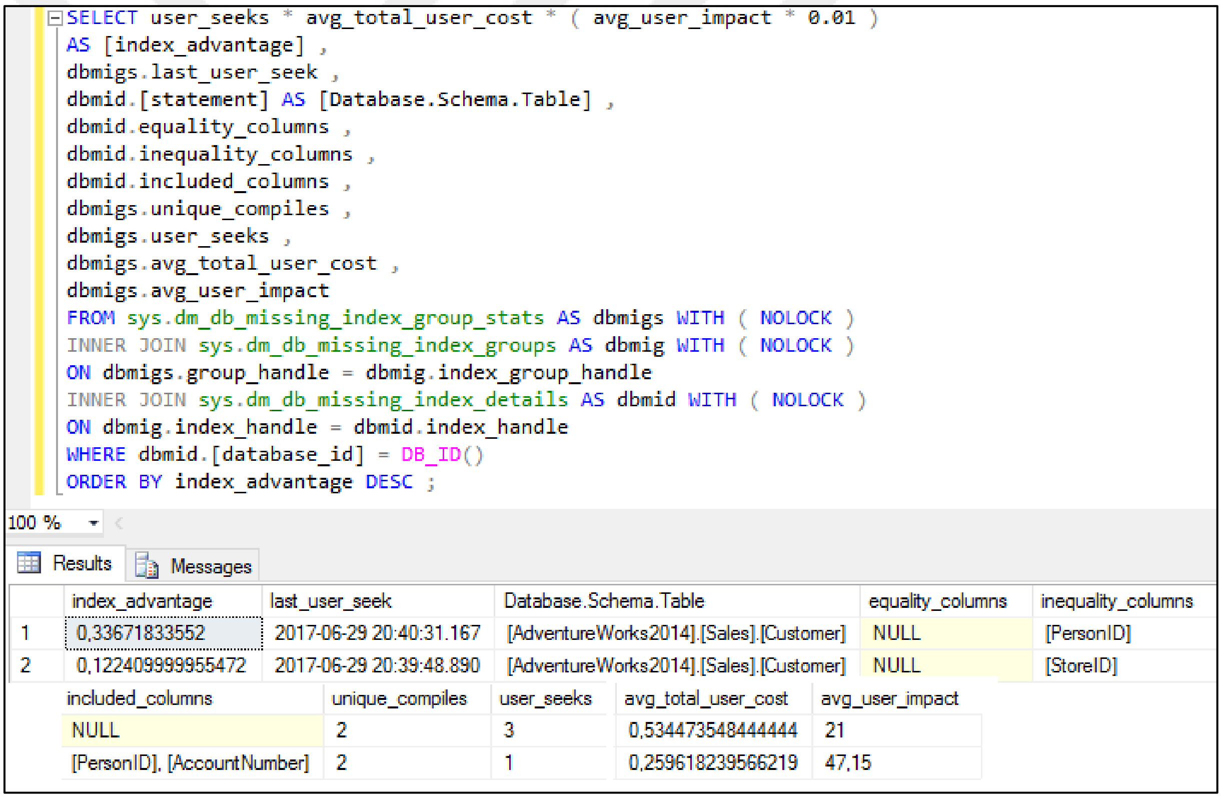
```sql
Select Top 20
ROUND(s.avg_total_user_cost * s.avg_user_impact
*(s.user_seeks + s.user_scans),0) as [Total Cost],
d.[statement] as [Table Name],
equality_columns,
inequality_columns,
included_columns
From sys.dm_db_missing_index_groups g
Inner Join sys.dm_db_missing_index_group_stats s
on s.group_handle = g.index_group_handle
Inner Join sys.dm_db_missing_index_details d
on d.index_handle = g.index_handle
Order by [Total Cost] Desc
```

100 %

Results Messages

	Total Cost	Table Name	equality_columns	inequality_columns	included_columns
1	34	[AdventureWorks2014].[Sales].[Customer]	NULL	[PersonID]	NULL
2	12	[AdventureWorks2014].[Sales].[Customer]	NULL	[StoreID]	[PersonID], [AccountNumber]

Figure 59. Missing Index Querying.



```sql
SELECT user_seeks * avg_total_user_cost * ( avg_user_impact * 0.01 )
AS [index_advantage] ,
dbmigs.last_user_seek ,
dbmid.[statement] AS [Database.Schema.Table] ,
dbmid.equality_columns ,
dbmid.inequality_columns ,
dbmid.included_columns ,
dbmigs.unique_compiles ,
dbmigs.user_seeks ,
dbmigs.avg_total_user_cost ,
dbmigs.avg_user_impact
FROM sys.dm_db_missing_index_group_stats AS dbmigs WITH ( NOLOCK )
INNER JOIN sys.dm_db_missing_index_groups AS dbmig WITH ( NOLOCK )
ON dbmigs.group_handle = dbmig.index_group_handle
INNER JOIN sys.dm_db_missing_index_details AS dbmid WITH ( NOLOCK )
ON dbmig.index_handle = dbmid.index_handle
WHERE dbmid.[database_id] = DB_ID()
ORDER BY index_advantage DESC ;
```

100 %

Results Messages

	index_advantage	last_user_seek	Database.Schema.Table	equality_columns	inequality_columns
1	0,33671833552	2017-06-29 20:40:31.167	[AdventureWorks2014].[Sales].[Customer]	NULL	[PersonID]
2	0,122409999955472	2017-06-29 20:39:48.890	[AdventureWorks2014].[Sales].[Customer]	NULL	[StoreID]

	included_columns	unique_compiles	user_seeks	avg_total_user_cost	avg_user_impact
1	NULL	2	3	0,534473548444444	21
2	[PersonID], [AccountNumber]	2	1	0,259618239566219	47,15

Figure 60. DMV Missing Index Querying.

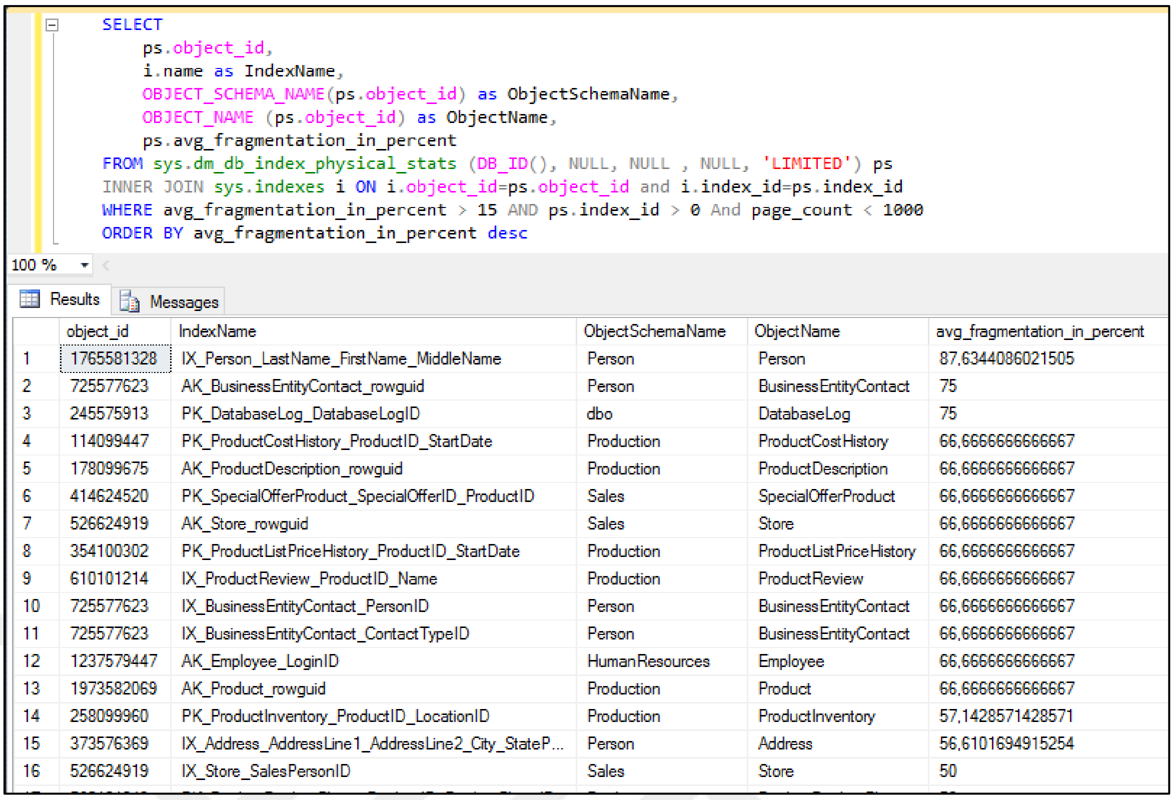
Finding missing index with DMV and DMF helps (Figure 59 and Figure 60). <u>Total cost column</u> shows the cost of missing index, <u>Table Name column</u> shows the table that needs to be applied to the index, <u>equality and inequality columns</u> shows key columns of index, <u>included column</u> shows the column that needs to adding to the index, <u>last user seek and scan column</u> the last time that seek and scan operations might have used the index., average total user cost column shows average cost saving for the queries that could have been

helped by the index in the group (Davidson & Ford, 2010), index advantage total benefit of recommended index. To optimize query performance users should check periodically missing index and their effects.

4.3 INDEX FRAGMENTATION

Index fragmentation is a serious performance problem. When index data deleted, inserted or updated index fragmentation can occur, when data is deleted gaps occur in data pages or updated or added data will fill the page and it effects logical order and physical order matching. All this operation directly effects disk I/O because when query wants reach the data it should dispose of time much more than expected so it effects performance negative way. To fix this user should specify the rate of fragmentation. To find the rate, users will use **sys.dm_db_index_phsical_stats** DMF. After detecting fragmentation, they have two options; Reorganize and Rebuild operations.

Rebuild operation is dropping the index and creating again. With dropping operation, fragmented index is completely removed. Reorganize operation is reorganizing the index pages that means reordering clustered or nonclustered index’s leaf level pages. To determine which operation will be apply on index users should see fragmentation rate if fragmentation rate is more than 30% Rebuild operation must be used (Figure 61).



```sql
SELECT
    ps.object_id,
    i.name as IndexName,
    OBJECT_SCHEMA_NAME(ps.object_id) as ObjectSchemaName,
    OBJECT_NAME (ps.object_id) as ObjectName,
    ps.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL , NULL, 'LIMITED') ps
INNER JOIN sys.indexes i ON i.object_id=ps.object_id and i.index_id=ps.index_id
WHERE avg_fragmentation_in_percent > 15 AND ps.index_id > 0 And page_count < 1000
ORDER BY avg_fragmentation_in_percent desc
```

100 %

Results | Messages

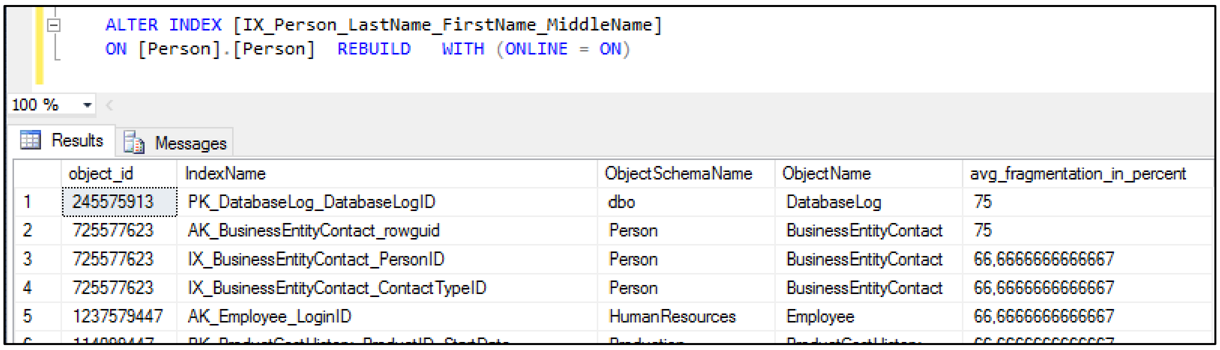
	object_id	IndexName	ObjectSchemaName	ObjectName	avg_fragmentation_in_percent
1	1765581328	IX_Person_LastName_FirstName_MiddleName	Person	Person	87,6344086021505
2	725577623	AK_BusinessEntityContact_rowguid	Person	BusinessEntityContact	75
3	245575913	PK_DatabaseLog_DatabaseLogID	dbo	DatabaseLog	75
4	114099447	PK_ProductCostHistory_ProductID_StartDate	Production	ProductCostHistory	66,6666666666667
5	178099675	AK_ProductDescription_rowguid	Production	ProductDescription	66,6666666666667
6	414624520	PK_SpecialOfferProduct_SpecialOfferID_ProductID	Sales	SpecialOfferProduct	66,6666666666667
7	526624919	AK_Store_rowguid	Sales	Store	66,6666666666667
8	354100302	PK_ProductListPriceHistory_ProductID_StartDate	Production	ProductListPriceHistory	66,6666666666667
9	610101214	IX_ProductReview_ProductID_Name	Production	ProductReview	66,6666666666667
10	725577623	IX_BusinessEntityContact_PersonID	Person	BusinessEntityContact	66,6666666666667
11	725577623	IX_BusinessEntityContact_ContactTypeID	Person	BusinessEntityContact	66,6666666666667
12	1237579447	AK_Employee_LoginID	HumanResources	Employee	66,6666666666667
13	1973582069	AK_Product_rowguid	Production	Product	66,6666666666667
14	258099960	PK_ProductInventory_ProductID_LocationID	Production	ProductInventory	57,1428571428571
15	373576369	IX_Address_AddressLine1_AddressLine2_City_StateP...	Person	Address	56,6101694915254
16	526624919	IX_Store_SalesPersonID	Sales	Store	50

Figure 61. Finding Fragmented Index.

Identifying and fixing one of the fragmented index from Figure 61. First fragmented index that has 87% fragmentation rate will be fixed. This index fragmentation rate is greater than 30% so users need to apply rebuild operation (Figure 62).

ALTER INDEX [IX_Person_LastName_FirstName_MiddleName]

ON [Person].[Person] REBUILD   WITH (ONLINE = ON)



```sql
ALTER INDEX [IX_Person_LastName_FirstName_MiddleName]
ON [Person].[Person]  REBUILD   WITH (ONLINE = ON)
```

100 %

Results | Messages

	object_id	IndexName	ObjectSchemaName	ObjectName	avg_fragmentation_in_percent
1	245575913	PK_DatabaseLog_DatabaseLogID	dbo	DatabaseLog	75
2	725577623	AK_BusinessEntityContact_rowguid	Person	BusinessEntityContact	75
3	725577623	IX_BusinessEntityContact_PersonID	Person	BusinessEntityContact	66,6666666666667
4	725577623	IX_BusinessEntityContact_ContactTypeID	Person	BusinessEntityContact	66,6666666666667
5	1237579447	AK_Employee_LoginID	HumanResources	Employee	66,6666666666667

Figure 62. Fixing Fragmentation.

As is seen, first index fragmentation is fixed with rebuild operation. To apply Reorganize one of the fragmented index that is smaller than 30% users should use this query:

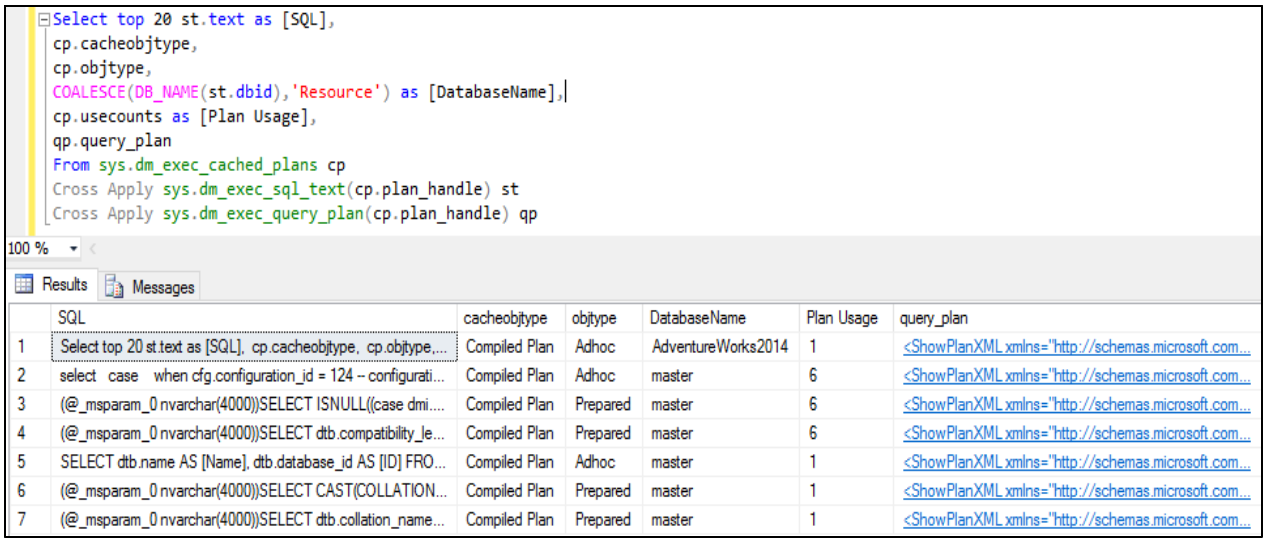
ALTER INDEX [AK_Product_Name]

ON [Production].[Product] REORGANIZE

After applied this operation users should create new index on this table and these columns.

4.4 EXECUTION PLAN ANALYSIS ON CACHE

As stated chapter 2 that before query execution, SQL Server create estimated execution plan estimates which index or procedure will be used for this query and this execution plan is saved on cached. If the query is executed again, this plan is used for this and its purpose is performance saving. However, if there is some difference in index or procedure SQL Server's used execution plan will be wrong it effects performance negative way. So, this execution plans on the cached should be checked and it can be done with these **sys.dm_db_cached_plans** DMV, **sys.dm_exec_sql_text**, **sys.dm_exec_query_plan** DMFs (Figure 63). Plan Usage is number of execution of execution plan. Users can control the execution plan when they do any changes on index or procedure.



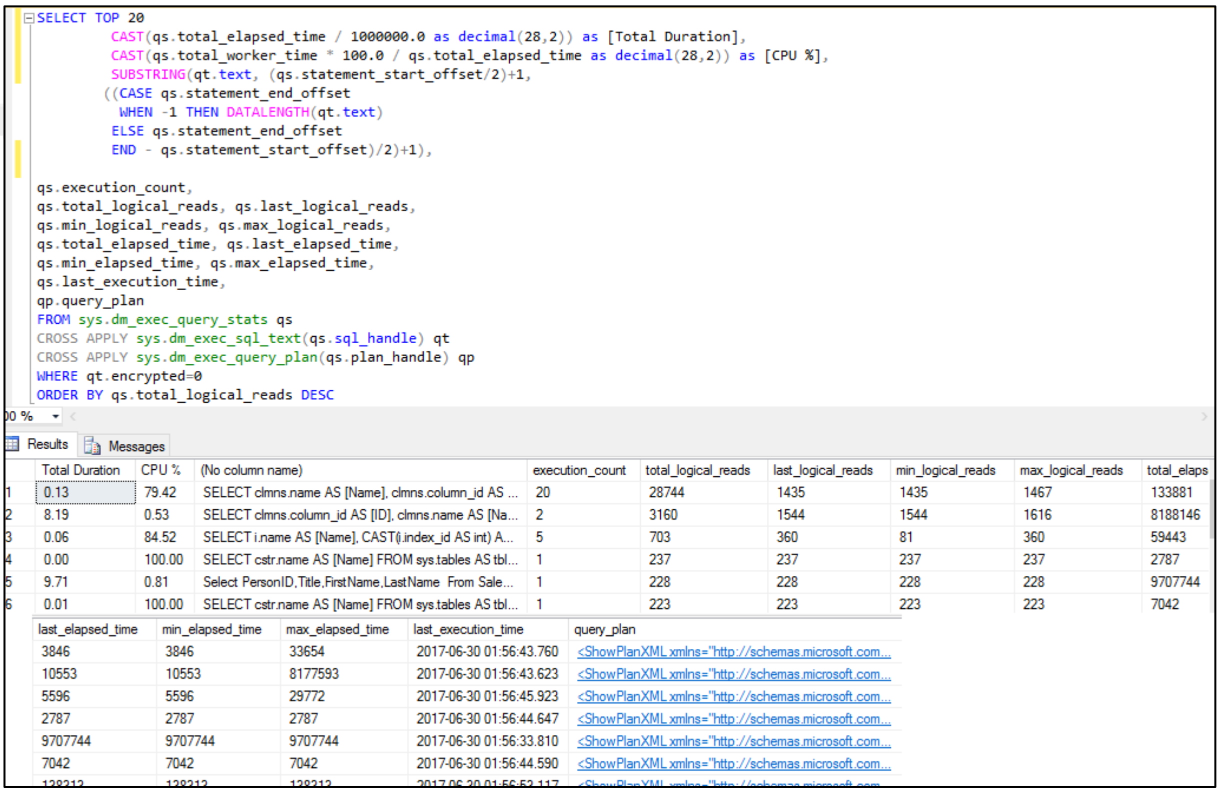
```
Select top 20 st.text as [SQL],
cp.cacheobjtype,
cp.objtype,
COALESCE(DB_NAME(st.dbid),'Resource') as [DatabaseName],
cp.usecounts as [Plan Usage],
qp.query_plan
From sys.dm_exec_cached_plans cp
Cross Apply sys.dm_exec_sql_text(cp.plan_handle) st
Cross Apply sys.dm_exec_query_plan(cp.plan_handle) qp
```

	SQL	cacheobjtype	objtype	DatabaseName	Plan Usage	query_plan
1	Select top 20 st.text as [SQL], cp.cacheobjtype, cp.objtype,...	Compiled Plan	Adhoc	AdventureWorks2014	1	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	select case when cfg.configuration_id = 124 -- configurati...	Compiled Plan	Adhoc	master	6	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	(@_msparam_0 nvarchar(4000))SELECT ISNULL((case dmi....	Compiled Plan	Prepared	master	6	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	(@_msparam_0 nvarchar(4000))SELECT dtb.compatibility_le...	Compiled Plan	Prepared	master	6	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	SELECT dtb.name AS [Name], dtb.database_id AS [ID] FRO...	Compiled Plan	Adhoc	master	1	<ShowPlanXML xmlns="http://schemas.microsoft.com...
6	(@_msparam_0 nvarchar(4000))SELECT CAST(COLLATION...	Compiled Plan	Prepared	master	1	<ShowPlanXML xmlns="http://schemas.microsoft.com...
7	(@_msparam_0 nvarchar(4000))SELECT dtb.collation_name...	Compiled Plan	Prepared	master	1	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Figure 63. Execution Plan Analysis on Cached.

4.5 FINDING TOP N MOST EXPENSIVE QUERIES

Expensive queries mean SQL Server spends more time to execute these queries. Users can reach same expensive queries with using Activity monitor section. Now **sys.dm_exec_query_stats** DMV and **sys.dm_exec_sql_text** and **sys.dm_exec_query_plan** DMFs will be used to find and interfere the queries (Figure 64).



```sql
SELECT TOP 20
         CAST(qs.total_elapsed_time / 1000000.0 as decimal(28,2)) as [Total Duration],
         CAST(qs.total_worker_time * 100.0 / qs.total_elapsed_time as decimal(28,2)) as [CPU %],
         SUBSTRING(qt.text, (qs.statement_start_offset/2)+1,
        ((CASE qs.statement_end_offset
          WHEN -1 THEN DATALENGTH(qt.text)
         ELSE qs.statement_end_offset
         END - qs.statement_start_offset)/2)+1),

qs.execution_count,
qs.total_logical_reads, qs.last_logical_reads,
qs.min_logical_reads, qs.max_logical_reads,
qs.total_elapsed_time, qs.last_elapsed_time,
qs.min_elapsed_time, qs.max_elapsed_time,
qs.last_execution_time,
qp.query_plan
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
WHERE qt.encrypted=0
ORDER BY qs.total_logical_reads DESC
```

	Total Duration	CPU %	(No column name)	execution_count	total_logical_reads	last_logical_reads	min_logical_reads	max_logical_reads	total_elaps
1	0.13	79.42	SELECT clmns.name AS [Name], clmns.column_id AS ...	20	28744	1435	1435	1467	133881
2	8.19	0.53	SELECT clmns.column_id AS [ID], clmns.name AS [Na...	2	3160	1544	1544	1616	8188146
3	0.06	84.52	SELECT i.name AS [Name], CAST(i.index_id AS int) A...	5	703	360	81	360	59443
4	0.00	100.00	SELECT cstr.name AS [Name] FROM sys.tables AS tbl...	1	237	237	237	237	2787
5	9.71	0.81	Select PersonID,Title,FirstName,LastName From Sale...	1	228	228	228	228	9707744
6	0.01	100.00	SELECT cstr.name AS [Name] FROM sys.tables AS tbl...	1	223	223	223	223	7042

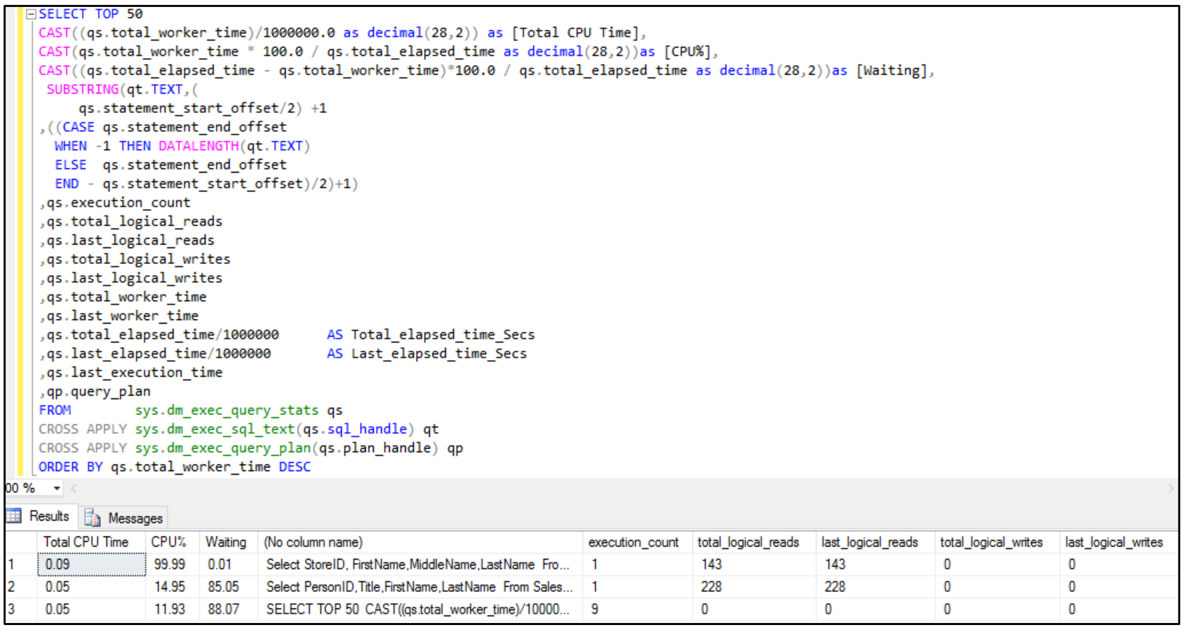
last_elapsed_time	min_elapsed_time	max_elapsed_time	last_execution_time	query_plan
3846	3846	33654	2017-06-30 01:56:43.760	<ShowPlanXML xmlns="http://schemas.microsoft.com...
10553	10553	8177593	2017-06-30 01:56:43.623	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5596	5596	29772	2017-06-30 01:56:45.923	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2787	2787	2787	2017-06-30 01:56:44.647	<ShowPlanXML xmlns="http://schemas.microsoft.com...
9707744	9707744	9707744	2017-06-30 01:56:33.810	<ShowPlanXML xmlns="http://schemas.microsoft.com...
7042	7042	7042	2017-06-30 01:56:44.590	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Figure 64. Most Expensive Queries.

Total duration column shows total execution time, CPU% column shows time that was spent to execute. With using these information users can detect most expensive queries and examine these queries to decrease execution times and increase queries' performance.

4.6 THE MOST CPU CONSUMED QUERIES

CPU is the most important system resource for SQL Server. When CPU spend more time to one query execution, server performance will be effected badly. For this reason, users need to control queries, which are consume CPU time more than expected. To identify these queries **sys.dm_exec_query_stats** DMV, **sys.dm_exec_sql_text** and **sys.dm_exec_query_plan** DMFs will be used (Figure 65).



```
SELECT TOP 50
CAST((qs.total_worker_time)/1000000.0 as decimal(28,2)) as [Total CPU Time],
CAST(qs.total_worker_time * 100.0 / qs.total_elapsed_time as decimal(28,2))as [CPU%],
CAST((qs.total_elapsed_time - qs.total_worker_time)*100.0 / qs.total_elapsed_time as decimal(28,2))as [Waiting],
 SUBSTRING(qt.TEXT,(
     qs.statement_start_offset/2) +1
,((CASE qs.statement_end_offset
  WHEN -1 THEN DATALENGTH(qt.TEXT)
  ELSE  qs.statement_end_offset
  END - qs.statement_start_offset)/2)+1)
,qs.execution_count
,qs.total_logical_reads
,qs.last_logical_reads
,qs.total_logical_writes
,qs.last_logical_writes
,qs.total_worker_time
,qs.last_worker_time
,qs.total_elapsed_time/1000000      AS Total_elapsed_time_Secs
,qs.last_elapsed_time/1000000       AS Last_elapsed_time_Secs
,qs.last_execution_time
,qp.query_plan
FROM        sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
ORDER BY qs.total_worker_time DESC
```

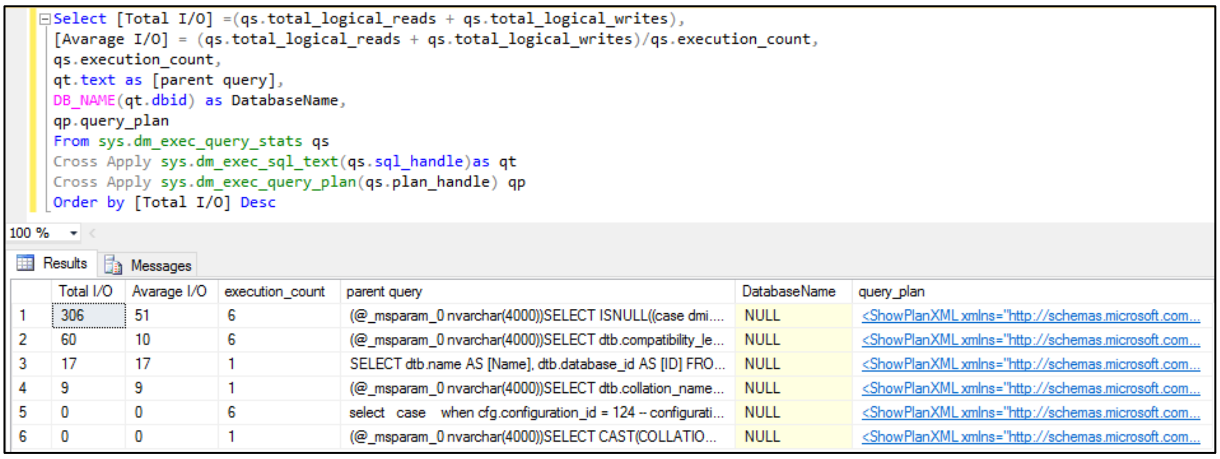
	Total CPU Time	CPU%	Waiting	(No column name)	execution_count	total_logical_reads	last_logical_reads	total_logical_writes	last_logical_writes
1	0.09	99.99	0.01	Select StoreID, FirstName,MiddleName,LastName Fro...	1	143	143	0	0
2	0.05	14.95	85.05	Select PersonID,Title,FirstName,LastName From Sales...	1	228	228	0	0
3	0.05	11.93	88.07	SELECT TOP 50 CAST((qs.total_worker_time)/10000...	9	0	0	0	0

Figure 65. CPU Consumed Queries.

Total CPU time column shows execution time on CPU, CPU percentage column shows percentage of CPU place. With controlling these values, users can make some necessary changes on queries to improve the query performance. It should not forgotten improving query performance means improving database performance.

4.7 DISK I/O BOTTLENECK

Disk I/O checking is important for performance measuring, if problem occurs on reading and writing processes that will probably effect the CPU and cached. So, to improve the performance of query users need to check disk I/O amount and detect the most I/O perform queries. To detect these queries sys.dm_exec_query_stats DMV, sys.dm_exec_sql_text and sys.dm_exec_query_plan DMFs are used (Figure 66).



```
Select [Total I/O] =(qs.total_logical_reads + qs.total_logical_writes),
[Avarage I/O] = (qs.total_logical_reads + qs.total_logical_writes)/qs.execution_count,
qs.execution_count,
qt.text as [parent query],
DB_NAME(qt.dbid) as DatabaseName,
qp.query_plan
From sys.dm_exec_query_stats qs
Cross Apply sys.dm_exec_sql_text(qs.sql_handle)as qt
Cross Apply sys.dm_exec_query_plan(qs.plan_handle) qp
Order by [Total I/O] Desc
```

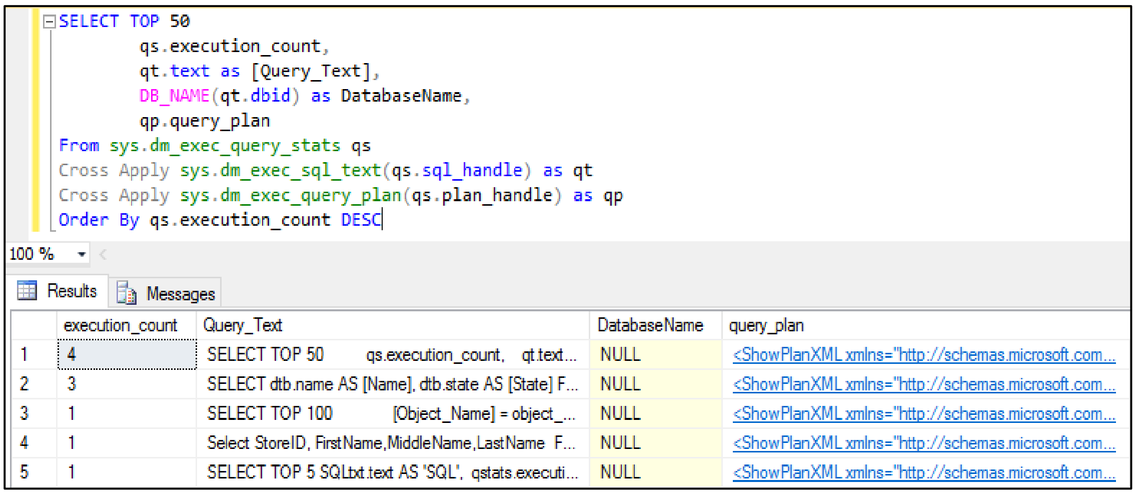
	Total I/O	Avarage I/O	execution_count	parent query	DatabaseName	query_plan
1	306	51	6	(@_msparam_0 nvarchar(4000))SELECT ISNULL((case dmi....	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	60	10	6	(@_msparam_0 nvarchar(4000))SELECT dtb.compatibility_le...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	17	17	1	SELECT dtb.name AS [Name], dtb.database_id AS [ID] FRO...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	9	9	1	(@_msparam_0 nvarchar(4000))SELECT dtb.collation_name...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	0	0	6	select case when cfg.configuration_id = 124 -- configurati...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
6	0	0	1	(@_msparam_0 nvarchar(4000))SELECT CAST(COLLATIO...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Figure 66. The most I/O Perform Queries.

Users can see the most I/O perform queries from this result, as a solution to minimize I/O using index on these queries (if relevant to use) will be lifesaving. As mentioned chapter 2, reading and writing data with index seek is easier for server. Finding query result is not exhausted with index.

4.8 THE MOST EXECUTED QUERIES

All Servers have most executed queries, maintenance of these queries is quite important. Most used query performance is important for users, these queries performances effect workflows. Thus, users need to control these queries periodically with **sys.dm_exec_query_stats** DMV, **sys.dm_exec_sql_text** and **sys.dm_exec_query_plan** DMFs (Figure 67). After finding most used queries, it can be used SQL Server tools to analyze them. After analyzing, queries needs specified and applied to speed up queries.



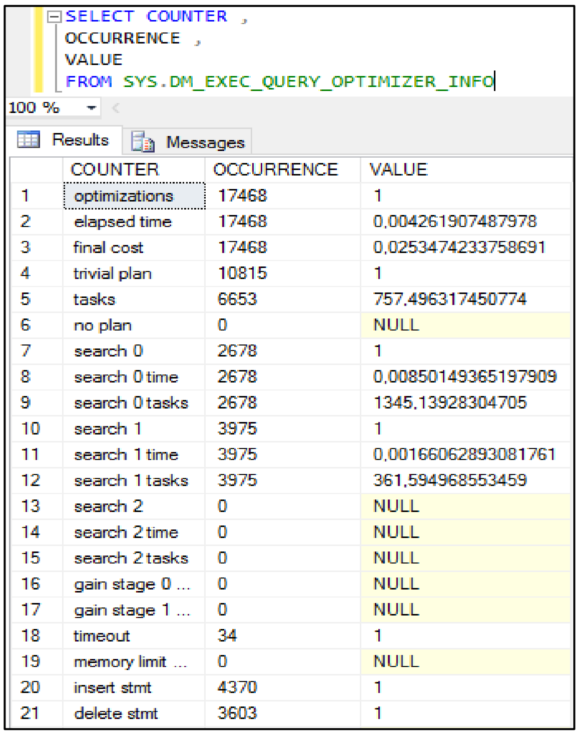
```
SELECT TOP 50
        qs.execution_count,
        qt.text as [Query_Text],
        DB_NAME(qt.dbid) as DatabaseName,
        qp.query_plan
From sys.dm_exec_query_stats qs
Cross Apply sys.dm_exec_sql_text(qs.sql_handle) as qt
Cross Apply sys.dm_exec_query_plan(qs.plan_handle) as qp
Order By qs.execution_count DESC
```

	execution_count	Query_Text	DatabaseName	query_plan
1	4	SELECT TOP 50 qs.execution_count, qt.text...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	3	SELECT dtb.name AS [Name], dtb.state AS [State] F...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	1	SELECT TOP 100 [Object_Name] = object_...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	1	Select StoreID, FirstName,MiddleName,LastName F...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	1	SELECT TOP 5 SQLtxt.text AS 'SQL', qstats.executi...	NULL	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Figure 67. The Most Executed Queries List.

4.9 QUERY OPTIMIZER

To get information about query optimizer users can use **sys.dm_exec_query_optimizer_info** DMV. This DMV is an aggregation of the optimization events over time (Fitchey, 2014). It tracks all optimization performs. Users can determine query performance situation with checking this DMV (Figure 68). This output includes all optimization performance information of optimizer.



```
SELECT COUNTER ,
OCCURRENCE ,
VALUE
FROM SYS.DM_EXEC_QUERY_OPTIMIZER_INFO
```

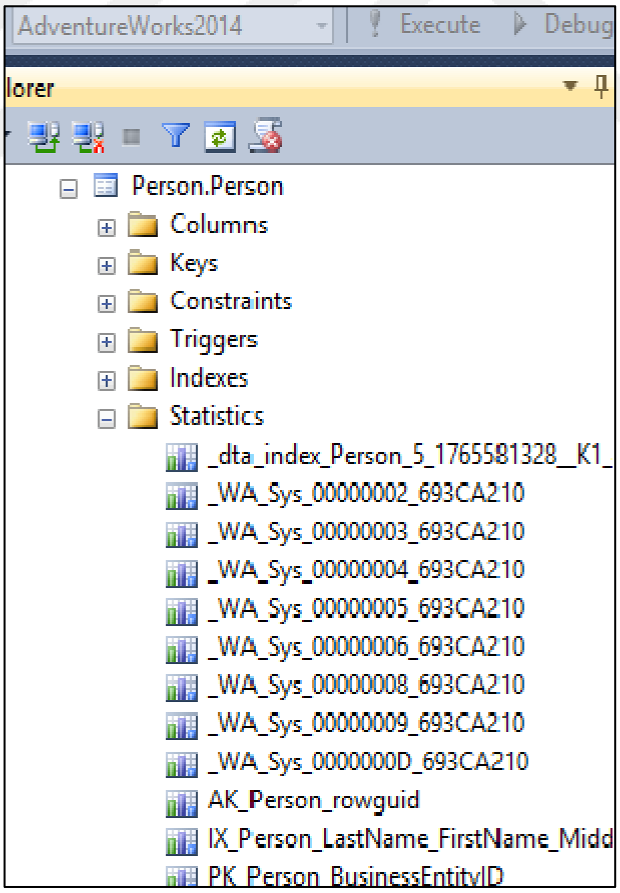
	COUNTER	OCCURRENCE	VALUE
1	optimizations	17468	1
2	elapsed time	17468	0,004261907487978
3	final cost	17468	0,0253474233758691
4	trivial plan	10815	1
5	tasks	6653	757,496317450774
6	no plan	0	NULL
7	search 0	2678	1
8	search 0 time	2678	0,00850149365197909
9	search 0 tasks	2678	1345,13928304705
10	search 1	3975	1
11	search 1 time	3975	0,00166062893081761
12	search 1 tasks	3975	361,594968553459
13	search 2	0	NULL
14	search 2 time	0	NULL
15	search 2 tasks	0	NULL
16	gain stage 0 ...	0	NULL
17	gain stage 1 ...	0	NULL
18	timeout	34	1
19	memory limit ...	0	NULL
20	insert stmt	4370	1
21	delete stmt	3603	1

Figure 68. Optimizer Information.

CHAPTER FIVE

IMPORTANCE OF STATISTICS

Statistics are very important to optimize query performance. Query optimizer uses statistics; it uses statistics to create optimal query plan structure. With using these, optimizer creates estimated cost and plan, optimizer decides to use index seek or index scan operation with using statistical information. Statistical data designate the way of optimizer. Users can reach already existing statistical information with using SQL Server Management Studio and control them. Besides, to monitor and quantify the performance of query users need to observe some statistical values for that query.

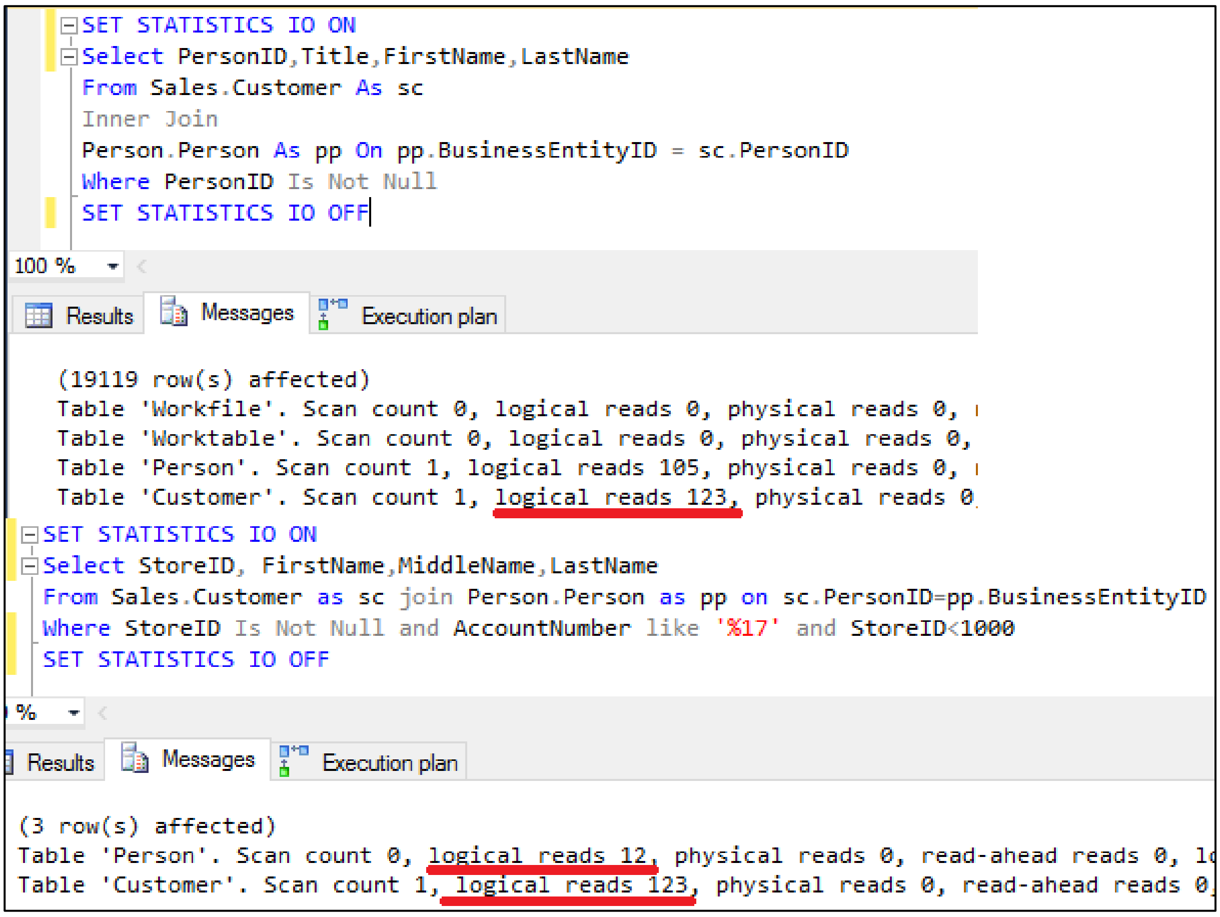


Person.Person
Columns
Keys
Constraints
Triggers
Indexes
Statistics
_dta_index_Person_5_1765581328__K1_
_WA_Sys_00000002_693CA210
_WA_Sys_00000003_693CA210
_WA_Sys_00000004_693CA210
_WA_Sys_00000005_693CA210
_WA_Sys_00000006_693CA210
_WA_Sys_00000008_693CA210
_WA_Sys_00000009_693CA210
_WA_Sys_0000000D_693CA210
AK_Person_rowguid
IX_Person_LastName_FirstName_Midd
PK_Person_BusinessEntityID

Figure 69. Statistics of Person.Person Table.

5.1 DISK I/O STATISTICS

These disk I/O statistics show users, the number of index seek or scan and logical read and write numbers. Logical read numbers will be small to get good performance from query. SET STATISTICS ON/OFF commands are used to see statistics. Closing statistics with OFF is important to do not occupy a place in cache. Statistical values on query four and query five are measured before using index (Figure 70);



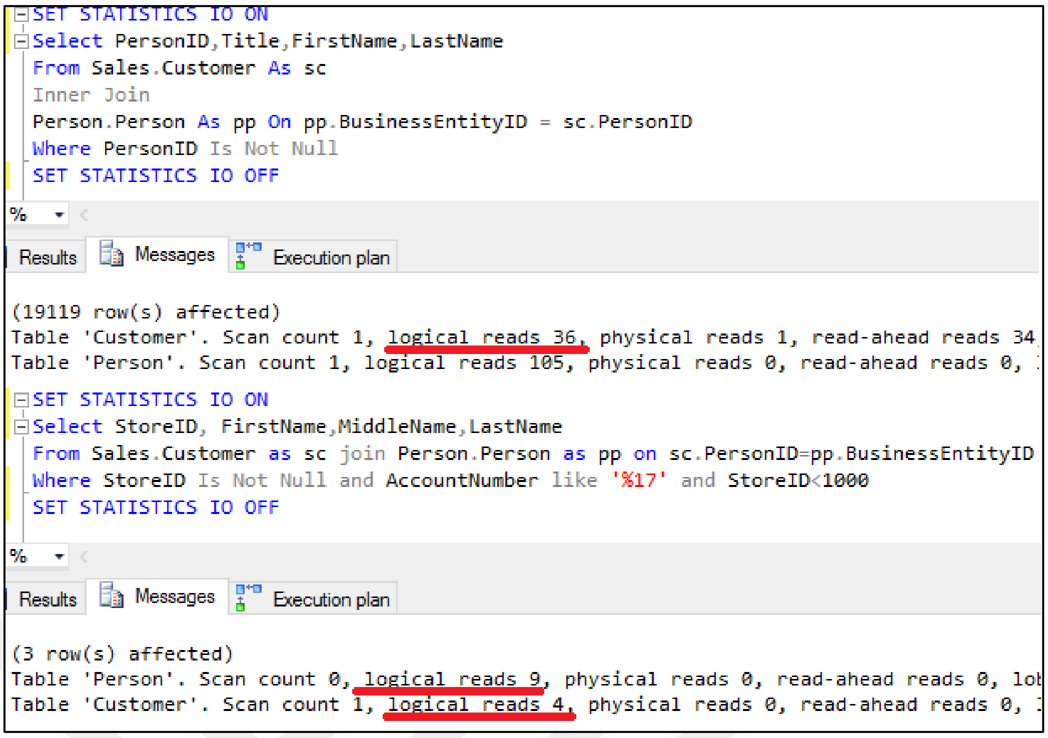
```
SET STATISTICS IO ON
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
SET STATISTICS IO OFF
```

```
(19119 row(s) affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0,
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
Table 'Person'. Scan count 1, logical reads 105, physical reads 0,
Table 'Customer'. Scan count 1, logical reads 123, physical reads 0
```

```
SET STATISTICS IO ON
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
SET STATISTICS IO OFF
```

```
(3 row(s) affected)
Table 'Person'. Scan count 0, logical reads 12, physical reads 0, read-ahead reads 0, l
Table 'Customer'. Scan count 1, logical reads 123, physical reads 0, read-ahead reads 0
```

Figure 70. Query Four and Query Five I/O Statistics before Index.


```
SET STATISTICS IO ON
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
SET STATISTICS IO OFF
```

Results | Messages | Execution plan

```
(19119 row(s) affected)
Table 'Customer'. Scan count 1, logical reads 36, physical reads 1, read-ahead reads 34
Table 'Person'. Scan count 1, logical reads 105, physical reads 0, read-ahead reads 0,
```

```
SET STATISTICS IO ON
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
SET STATISTICS IO OFF
```

Results | Messages | Execution plan

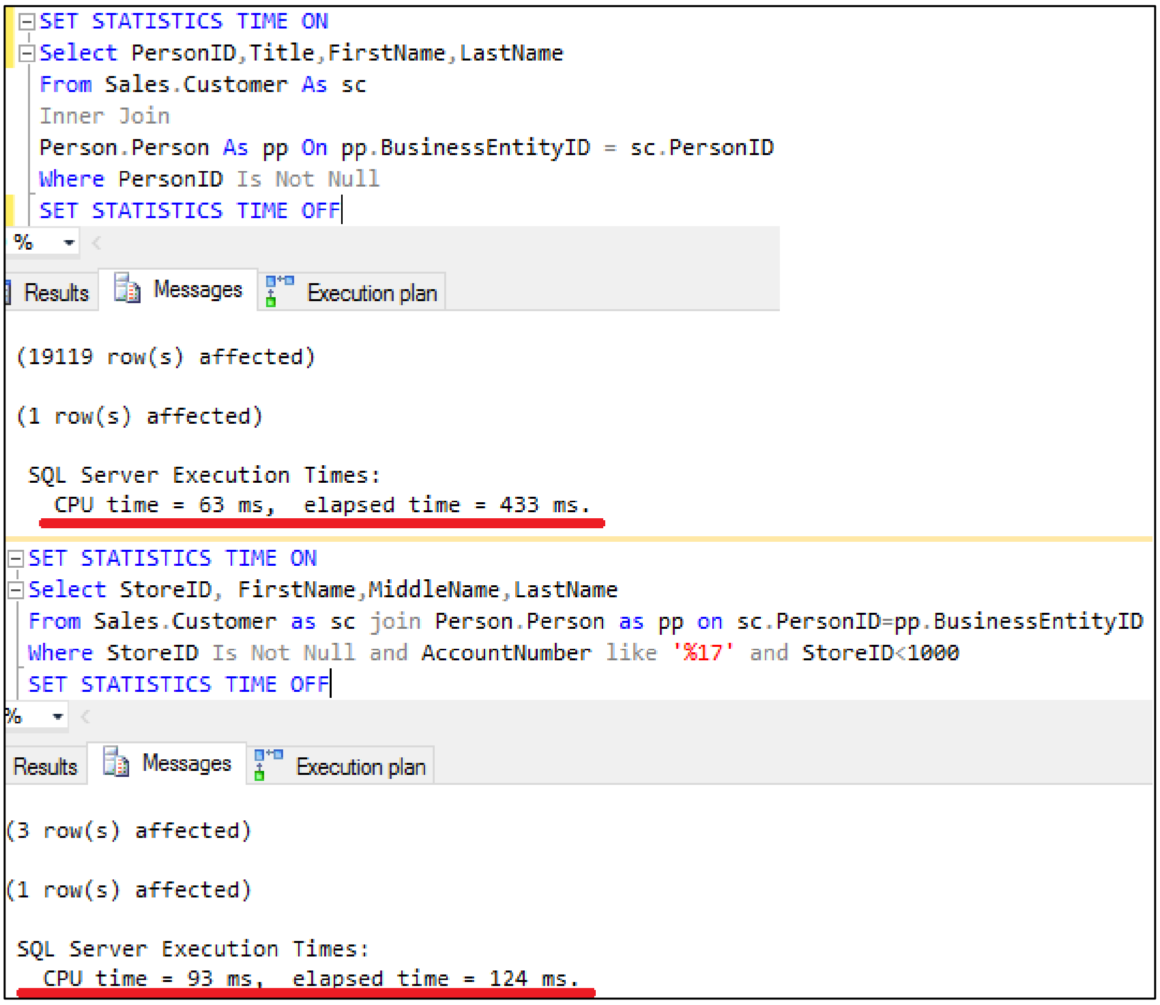
```
(3 row(s) affected)
Table 'Person'. Scan count 0, logical reads 9, physical reads 0, read-ahead reads 0, lo
Table 'Customer'. Scan count 1, logical reads 4, physical reads 0, read-ahead reads 0,
```

Figure 71. I/O Statistics after Index.

As is seen logical reads have high values Figure 70, the server spends a lot of effort for I / O operation. Index can be used to reduce this seek operation (Figure 71); After using necessary index Figure 71, query performance improved with reducing logical reads.

5.2 EXECUTION TIME STATISTICS

Execution time is important for query performance so CPU time managed before chapters. In this chapter, CPU time will be controlled with using statistics, which are SET STATISTICS TIME ON/OFF. The execution time of query four and five are going to be measured without applying index (Figure 72).



```
SET STATISTICS TIME ON
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
SET STATISTICS TIME OFF
```

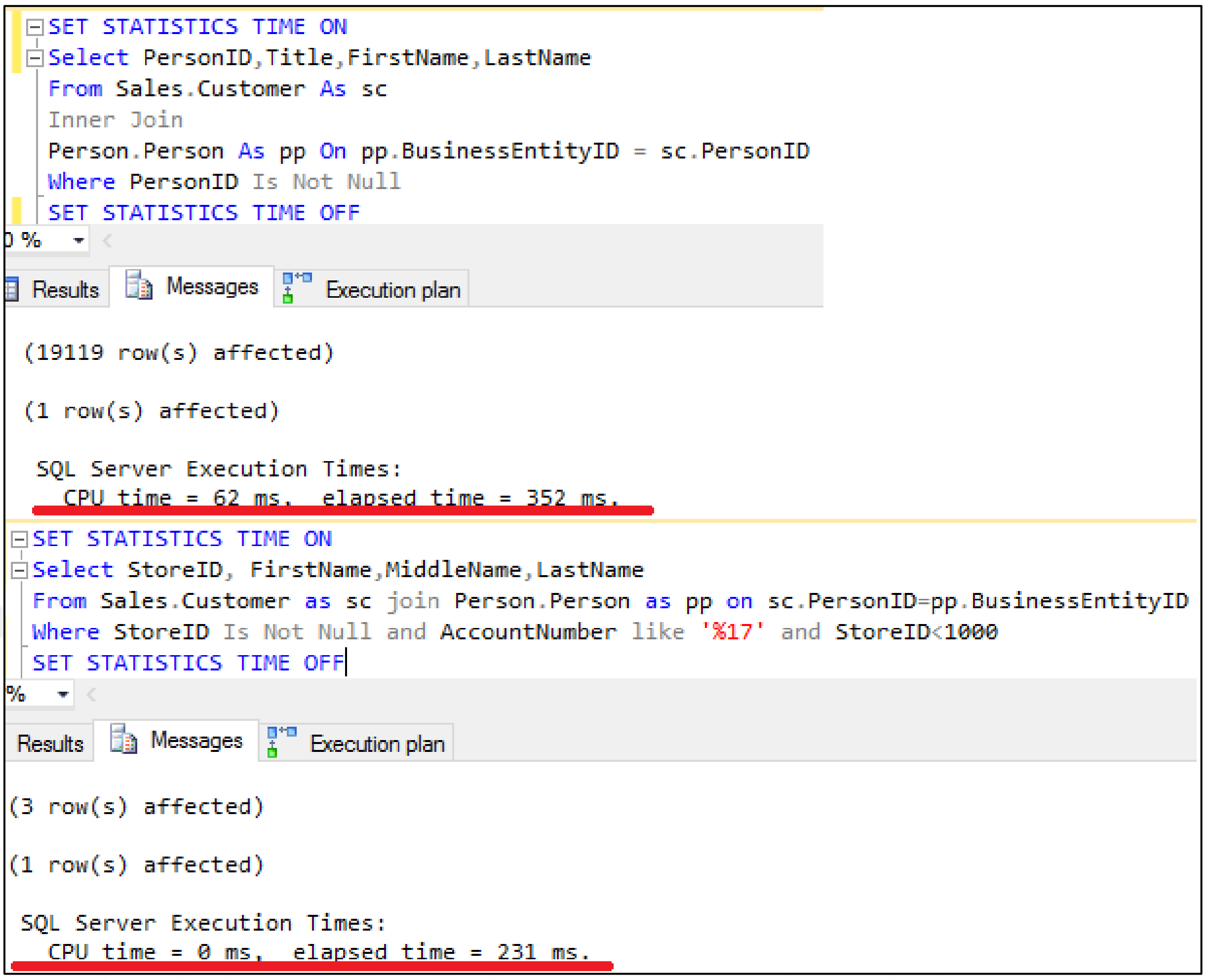
```
(19119 row(s) affected)

(1 row(s) affected)

 SQL Server Execution Times:
   CPU time = 63 ms,  elapsed time = 433 ms.
```

```
SET STATISTICS TIME ON
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
SET STATISTICS TIME OFF
```

```
(3 row(s) affected)

(1 row(s) affected)

 SQL Server Execution Times:
   CPU time = 93 ms,  elapsed time = 124 ms.
```

Figure 72. Execution Time Statistics before Index.

To performance improvement Figure72, CPU time needs to reduce. Necessary index can be applied on queries. After applied index CPU time and elapsed time decrease that shows the improvement of performance (Figure 73). So, it can be said that index which are used on these queries is useful for improvement.



```
SET STATISTICS TIME ON
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
SET STATISTICS TIME OFF
```

```
(19119 row(s) affected)

(1 row(s) affected)

 SQL Server Execution Times:
   CPU time = 62 ms,  elapsed time = 352 ms.
```

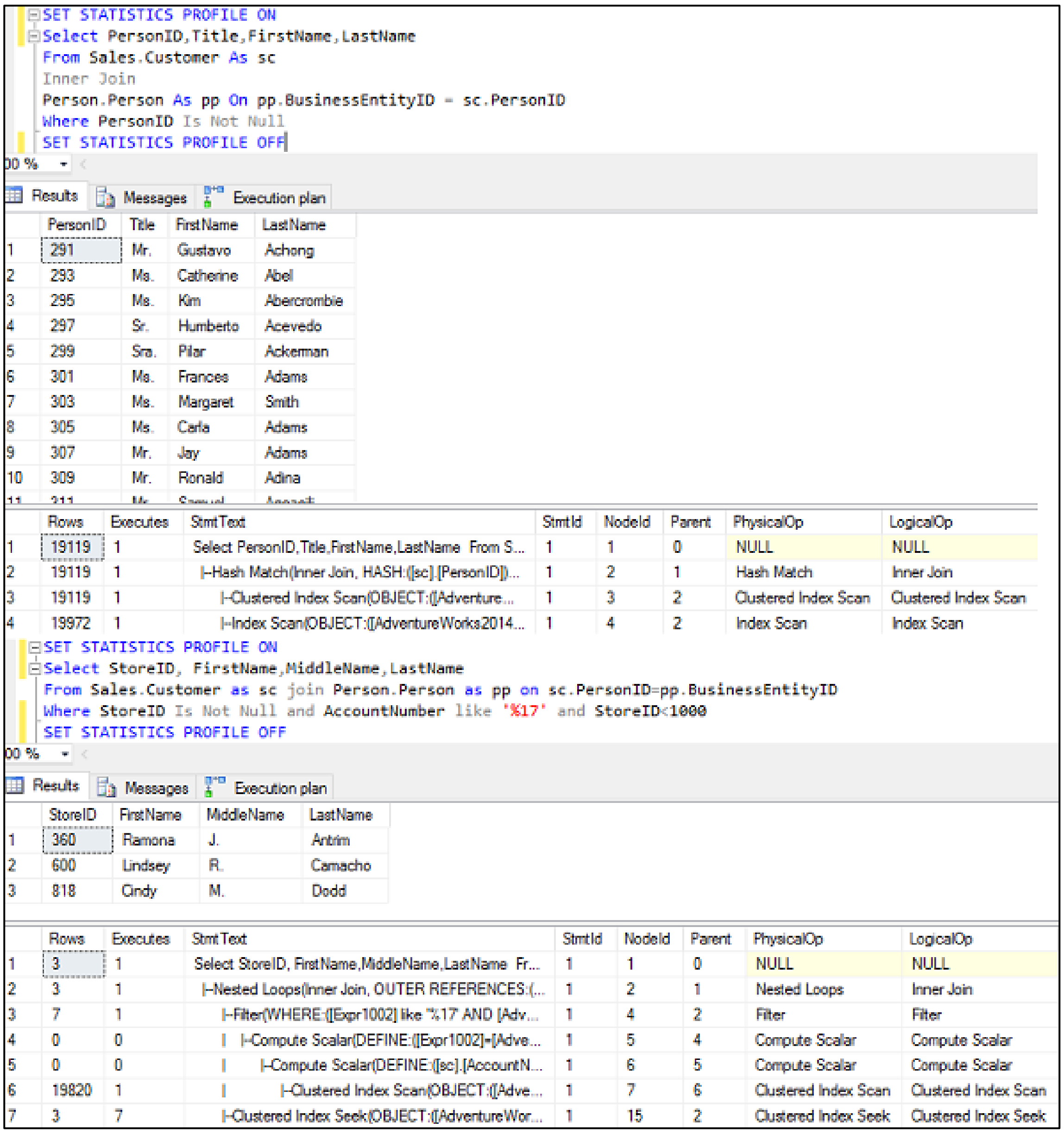
```
SET STATISTICS TIME ON
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
SET STATISTICS TIME OFF
```

```
(3 row(s) affected)

(1 row(s) affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 231 ms.
```

Figure 73. Execution Time Statistics after Index Applied.

5.3 MONITORING STATISTICS

With SET STATISTICS PROFILE ON/OFF command, it can monitor the query to get its necessity to improve performance (Figure 74).



```sql
SET STATISTICS PROFILE ON
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
SET STATISTICS PROFILE OFF
```

```
     PersonID  Title  FirstName  LastName
1    291       Mr.    Gustavo    Achong
2    293       Ms.    Catherine  Abel
3    295       Ms.    Kim        Abercrombie
4    297       Sr.    Humberto   Acevedo
5    299       Sra.   Pilar      Ackerman
6    301       Ms.    Frances    Adams
7    303       Ms.    Margaret   Smith
8    305       Ms.    Carla      Adams
9    307       Mr.    Jay        Adams
10   309       Mr.    Ronald     Adina
```

```
   Rows   Executes  StmtText                                              StmtId  NodeId  Parent  PhysicalOp            LogicalOp
1  19119  1         Select PersonID,Title,FirstName,LastName  From S...   1       1       0       NULL                  NULL
2  19119  1           |-Hash Match(Inner Join, HASH:([sc].[PersonID])...  1       2       1       Hash Match            Inner Join
3  19119  1              |-Clustered Index Scan(OBJECT:([Adventure...     1       3       2       Clustered Index Scan  Clustered Index Scan
4  19972  1              |-Index Scan(OBJECT:([AdventureWorks2014...      1       4       2       Index Scan            Index Scan
```

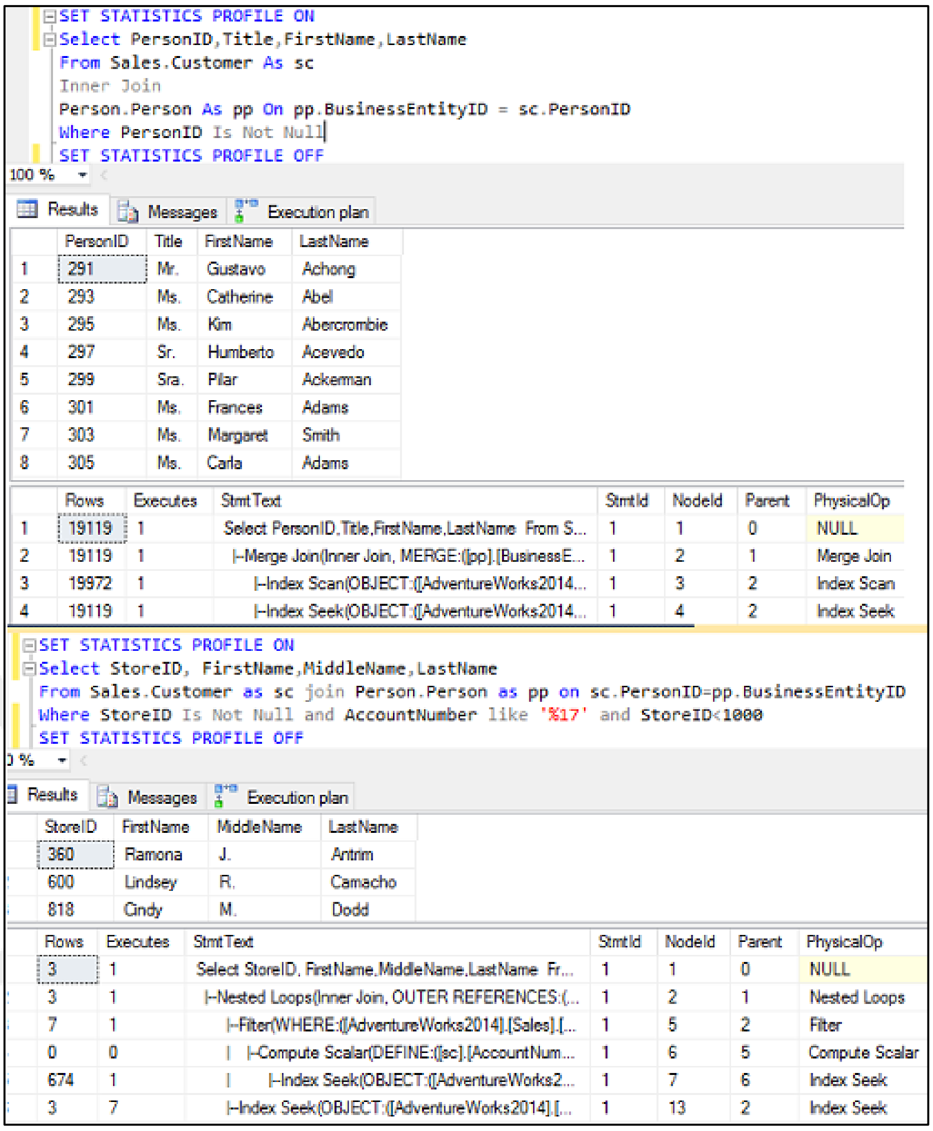
```sql
SET STATISTICS PROFILE ON
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
SET STATISTICS PROFILE OFF
```

```
   StoreID  FirstName  MiddleName  LastName
1  360      Ramona     J.          Antrim
2  600      Lindsey    R.          Camacho
3  818      Cindy      M.          Dodd
```

```
   Rows   Executes  StmtText                                              StmtId  NodeId  Parent  PhysicalOp            LogicalOp
1  3      1         Select StoreID, FirstName,MiddleName,LastName  Fr...  1       1       0       NULL                  NULL
2  3      1           |-Nested Loops(Inner Join, OUTER REFERENCES:(...    1       2       1       Nested Loops          Inner Join
3  7      1              |-Filter(WHERE:([Expr1002] like '%17' AND [Adv...  1       4       2       Filter                Filter
4  0      0              |    |-Compute Scalar(DEFINE:([Expr1002]=[Adve...  1     5       4       Compute Scalar        Compute Scalar
5  0      0              |         |-Compute Scalar(DEFINE:([sc].[AccountN...  1  6       5       Compute Scalar        Compute Scalar
6  19820  1              |              |-Clustered Index Scan(OBJECT:([Adve...  1  7     6       Clustered Index Scan  Clustered Index Scan
7  3      7              |-Clustered Index Seek(OBJECT:([AdventureWor...      1       15      2       Clustered Index Seek  Clustered Index Seek
```

Figure 74. Query Four and Five Without Index.

For first query physical operation columns and rows column can be checked, and to understand this statistics second query is more clarify, rows, nodeid, parent and physical operation columns' values can be examined and compared with Figure 75 values.



```sql
SET STATISTICS PROFILE ON
Select PersonID,Title,FirstName,LastName
From Sales.Customer As sc
Inner Join
Person.Person As pp On pp.BusinessEntityID = sc.PersonID
Where PersonID Is Not Null
SET STATISTICS PROFILE OFF
```

	PersonID	Title	FirstName	LastName
1	291	Mr.	Gustavo	Achong
2	293	Ms.	Catherine	Abel
3	295	Ms.	Kim	Abercrombie
4	297	Sr.	Humberto	Acevedo
5	299	Sra.	Pilar	Ackerman
6	301	Ms.	Frances	Adams
7	303	Ms.	Margaret	Smith
8	305	Ms.	Carla	Adams

	Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp
1	19119	1	Select PersonID,Title,FirstName,LastName From S...	1	1	0	NULL
2	19119	1	|--Merge Join(Inner Join, MERGE:([pp].[BusinessE...	1	2	1	Merge Join
3	19972	1	|--Index Scan(OBJECT:([AdventureWorks2014...	1	3	2	Index Scan
4	19119	1	|--Index Seek(OBJECT:([AdventureWorks2014...	1	4	2	Index Seek

```sql
SET STATISTICS PROFILE ON
Select StoreID, FirstName,MiddleName,LastName
From Sales.Customer as sc join Person.Person as pp on sc.PersonID=pp.BusinessEntityID
Where StoreID Is Not Null and AccountNumber like '%17' and StoreID<1000
SET STATISTICS PROFILE OFF
```

	StoreID	FirstName	MiddleName	LastName
	360	Ramona	J.	Antrim
	600	Lindsey	R.	Camacho
	818	Cindy	M.	Dodd

	Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp
	3	1	Select StoreID, FirstName,MiddleName,LastName Fr...	1	1	0	NULL
	3	1	|--Nested Loops(Inner Join, OUTER REFERENCES:(...	1	2	1	Nested Loops
	7	1	|--Filter(WHERE:([AdventureWorks2014].[Sales].[...	1	5	2	Filter
	0	0	| |--Compute Scalar(DEFINE:([sc].[AccountNum...	1	6	5	Compute Scalar
	674	1	| |--Index Seek(OBJECT:([AdventureWorks2...	1	7	6	Index Seek
	3	7	|--Index Seek(OBJECT:([AdventureWorks2014].[...	1	13	2	Index Seek

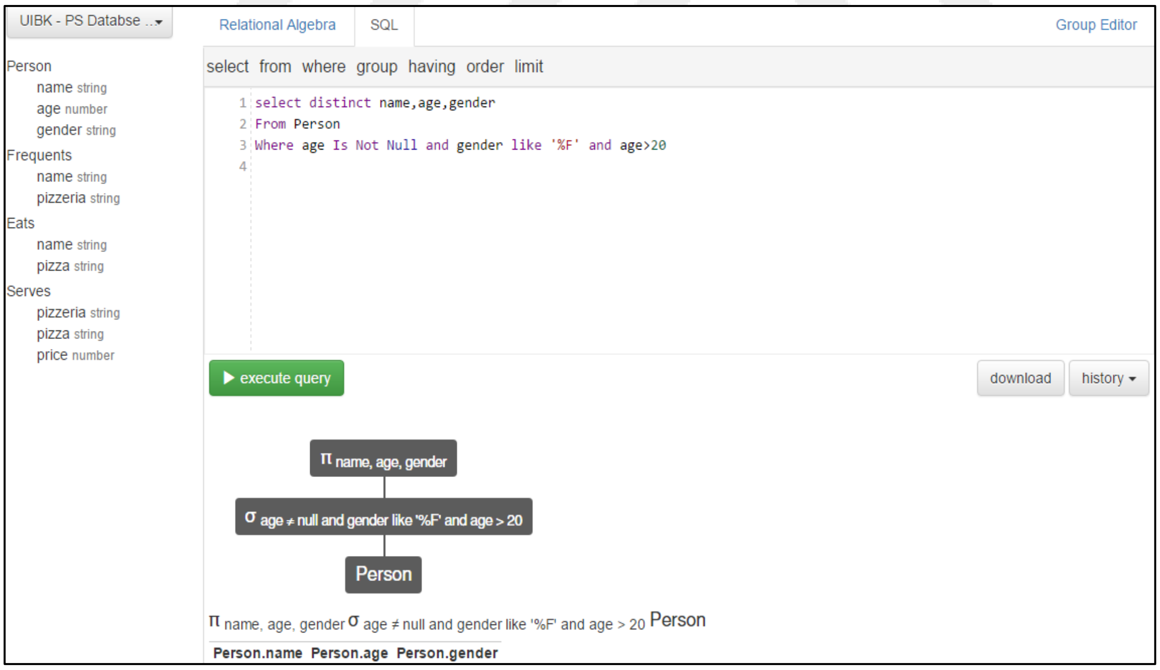
Figure 75. Query Four and Five With Index.

To measure improvement of query four users can check Physical Operation columns, it shows that, after using index, merge join, index scan and index seek operations are used for getting result, and Rows column shows that searching data number decreased 19972 to 19119 for applied index table. For second query Rows column shows that using index decrease searching data number 19820 to 674 for index applied table and index seek operation is used for improve the performance.

CHAPTER SIX
RELATED WORKS

In this chapter will be introduced other tools, which are do same things with SQL Server tools.

- **RelaX - Relational Algebra Calculator:** This calculator transforms relational algebra to SQL query and SQL query to relational algebra expression and shows query tree. It has own data users can learn relational algebra with use this data also, users can create their own database to use it. It can be used online (Figure 76).



Person
  name string
  age number
  gender string
Frequents
  name string
  pizzeria string
Eats
  name string
  pizza string
Serves
  pizzeria string
  pizza string
  price number

```
select distinct name,age,gender
From Person
Where age Is Not Null and gender like '%F' and age>20
```

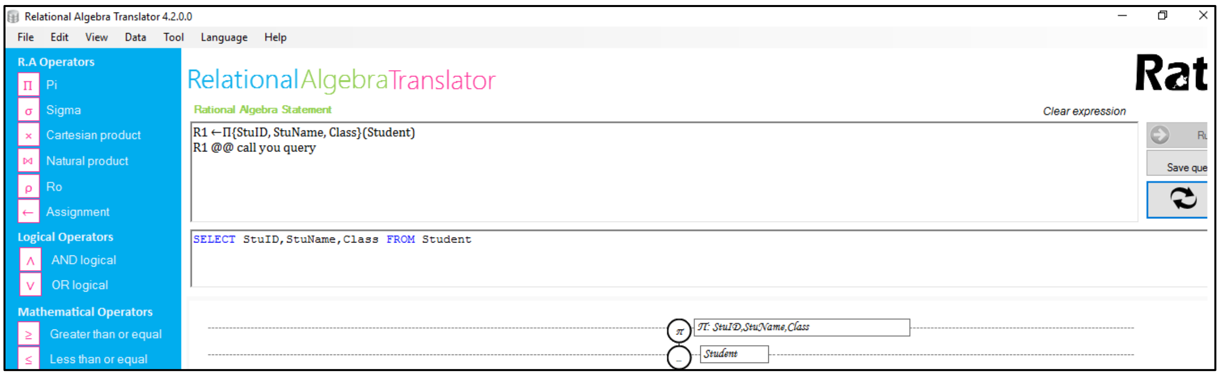
π name, age, gender σ age ≠ null and gender like '%F' and age > 20 Person

Person.name Person.age Person.gender

Figure 76. Display of RelaX.

- **RAT – Relational Algebra Translator:** With this tool, users can transform their relational algebra to SQL query. Thus, this tool shows query tree like

RelaX. It can be downloaded as free (Figure 77).



```
R1 ←Π{StuID, StuName, Class}(Student)
R1 @@ call you query
```

```
SELECT StuID,StuName,Class FROM Student
```

Figure 77. Display of RAT.

- **DATABASE MONITOR PRTG:** Users can get this database performance analyzer free at its own site. With this analyzer, user can monitor their database, no matter what databases are MySQL, Microsoft SQL, Oracle SQL, and PostgreSQL (Figure 78).

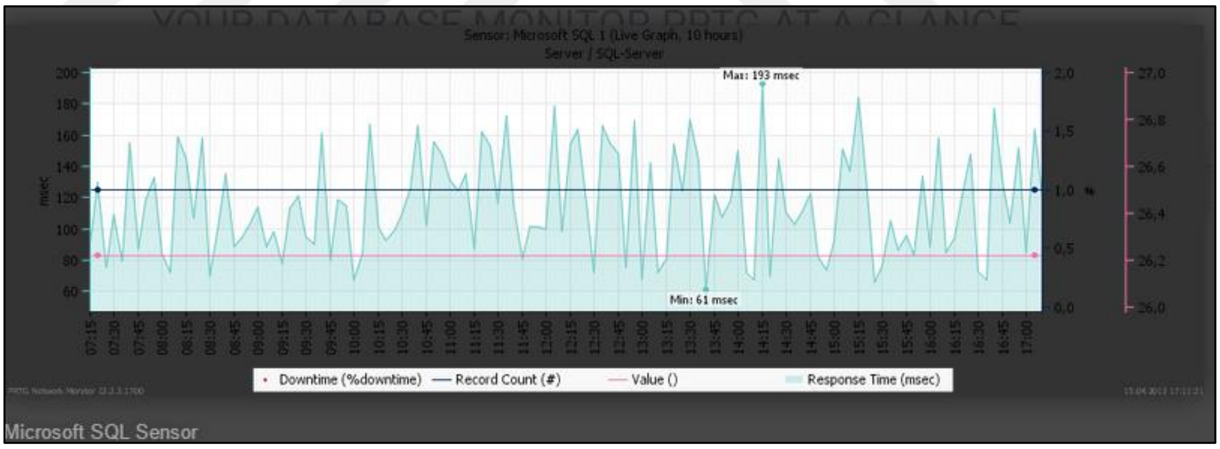
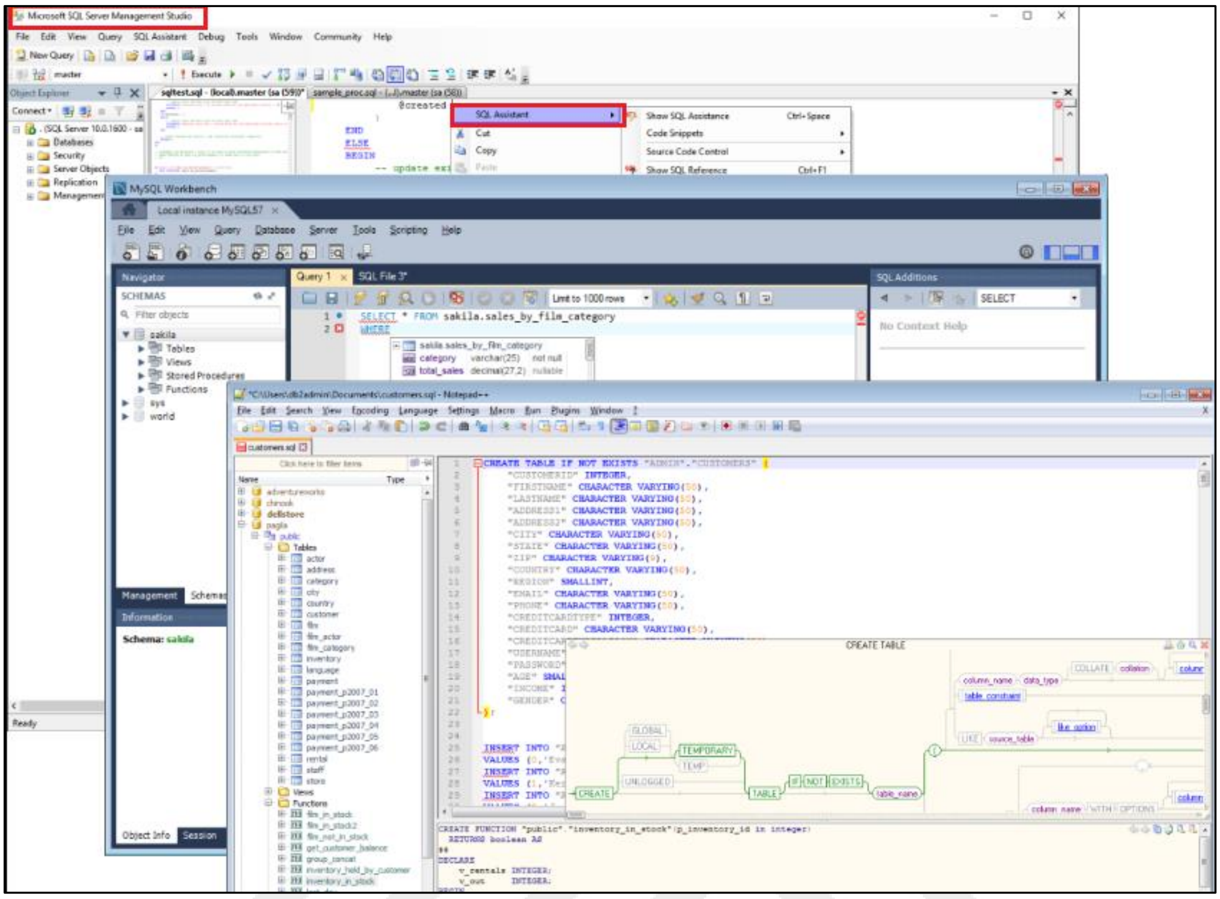


Figure 78. Display of MS SQL Sensor (www.paessler.com/database-monitoring).

- **SQL ASSISTANT:** It can be used with MS SQL Server, it has own performance measurement tools like SQL Server. Users can use it to improve the query and find differences between SQL Server solutions (Figure 79).



```
SELECT * FROM sakila.sales_by_film_category
WHERE
```

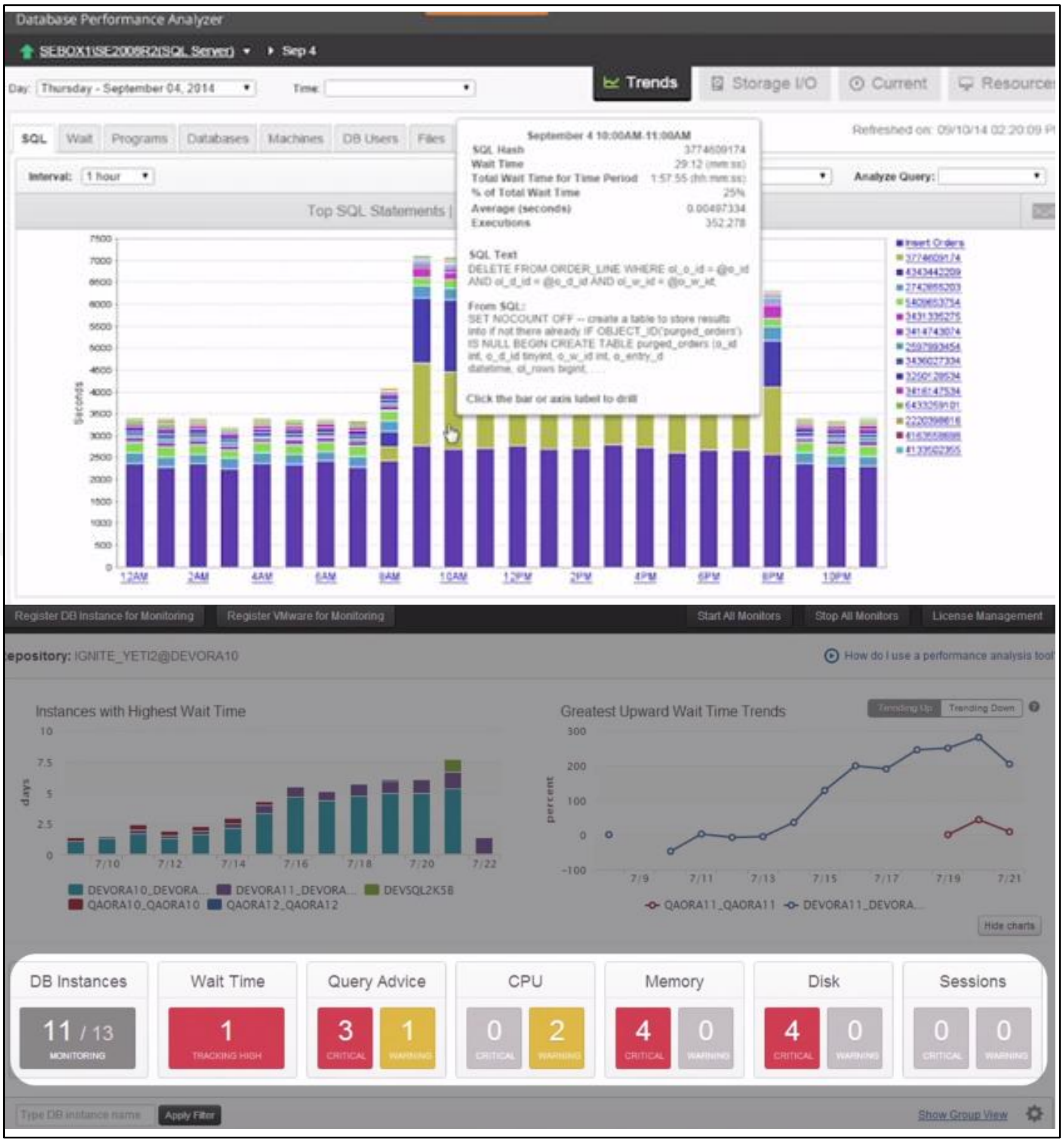
```
sakila.sales_by_film_category
category      varchar(25)    not null
total_sales   decimal(27,2)  nullable
```

```
CREATE TABLE IF NOT EXISTS "ADMIN"."CUSTOMERS" (
    "CUSTOMERID" INTEGER,
    "FIRSTNAME" CHARACTER VARYING(50),
    "LASTNAME" CHARACTER VARYING(50),
    "ADDRESS1" CHARACTER VARYING(50),
    "ADDRESS2" CHARACTER VARYING(50),
    "CITY" CHARACTER VARYING(50),
    "STATE" CHARACTER VARYING(50),
    "ZIP" CHARACTER VARYING(9),
    "COUNTRY" CHARACTER VARYING(50),
    "REGION" SMALLINT,
    "EMAIL" CHARACTER VARYING(50),
    "PHONE" CHARACTER VARYING(50),
    "CREDITCARDTYPE" INTEGER,
    "CREDITCARD" CHARACTER VARYING(50),
```

```
CREATE FUNCTION "public"."inventory_in_stock"(p_inventory_id in integer)
  RETURNS boolean AS
$$
DECLARE
    v_rentals INTEGER;
    v_out     INTEGER;
```

Figure 79. Display of SQL Assistant on Databases (www.softtreetech.com/sqlassist/index.htm).

- **SOLARWINDS DATABASE PERFORMANCE ANALYZER:** Performance analyzer is very useful for MS SQL Server performance tracking, with this analyzer users can watch their each query performance daily. They can collect their data from analyzer and fix the query and other events that effect the performance (Figure 80).



Repository: IGNITE_YETI2@DEVORA10

Refreshed on: 09/10/14 02:20:09 PM

September 4 10:00AM-11:00AM

SQL Hash	3774609174
Wait Time	29:12 (mm:ss)
Total Wait Time for Time Period	1:57:55 (hh:mm:ss)
% of Total Wait Time	25%
Average (seconds)	0.00497334
Executions	352,278

SQL Text

```
DELETE FROM ORDER_LINE WHERE ol_o_id = @o_id
AND ol_d_id = @o_d_id AND ol_w_id = @o_w_id;
```

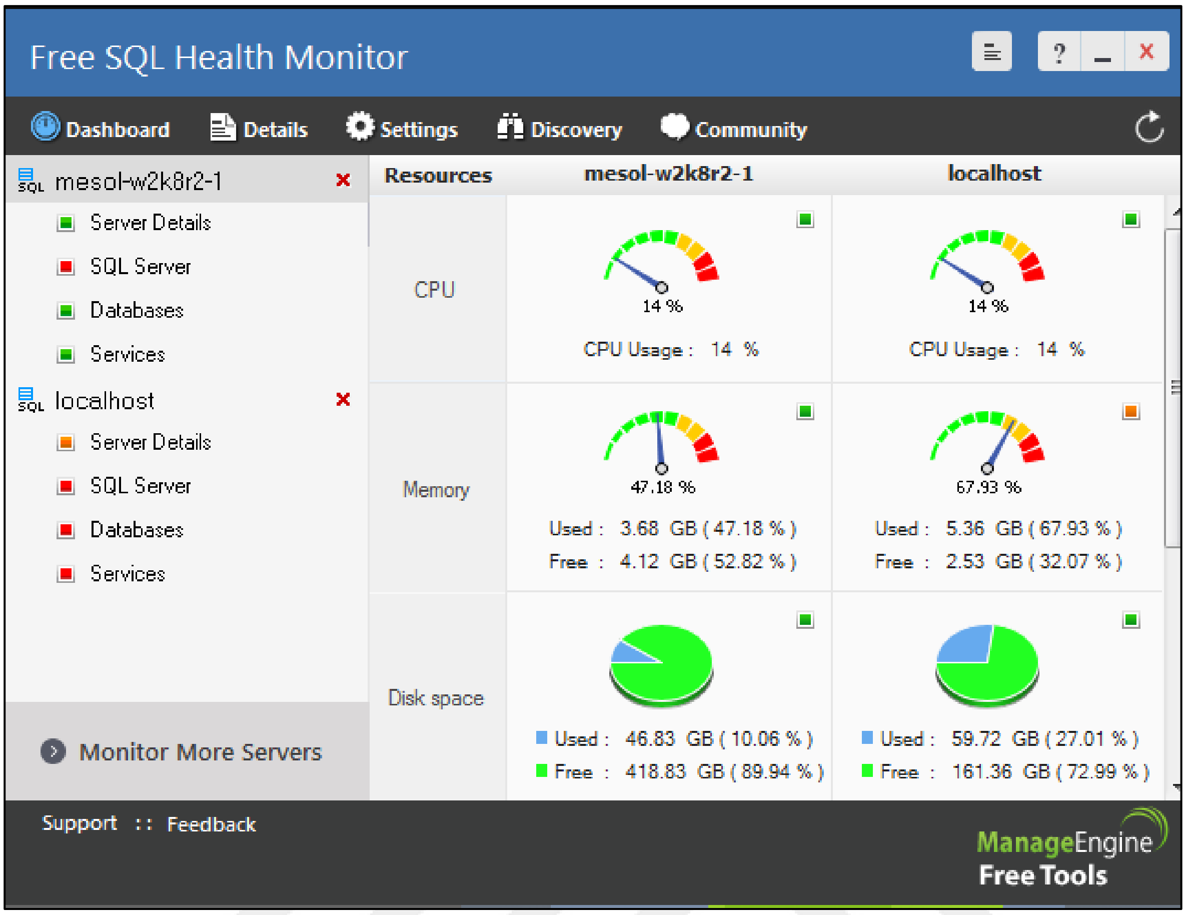
From SQL:

```
SET NOCOUNT OFF -- create a table to store results
into if not there already IF OBJECT_ID('purged_orders')
IS NULL BEGIN CREATE TABLE purged_orders (o_id
int, o_d_id tinyint, o_w_id int, o_entry_d
datetime, ol_rows bigint, ....
```

Click the bar or axis label to drill

Figure 80. Display of SolarWinds (www.solarwinds.com/database-performance-analyzer).

- **FREE SQL PERFORMANCE MONITORING TOOL:** With this tool users can monitor CPU, memory, disk and locks. They can watch their server performance here and they can determine the operation, which one is healthier for their query (Figure 81).


Free SQL Health Monitor

Dashboard Details Settings Discovery Community

mesol-w2k8r2-1
- 🟩 Server Details
- 🟥 SQL Server
- 🟩 Databases
- 🟩 Services

localhost
- 🟧 Server Details
- 🟥 SQL Server
- 🟥 Databases
- 🟥 Services

Monitor More Servers

Resources	mesol-w2k8r2-1	localhost
CPU	14 % CPU Usage : 14 %	14 % CPU Usage : 14 %
Memory	47.18 % Used : 3.68 GB (47.18 %) Free : 4.12 GB (52.82 %)	67.93 % Used : 5.36 GB (67.93 %) Free : 2.53 GB (32.07 %)
Disk space	Used : 46.83 GB (10.06 %) Free : 418.83 GB (89.94 %)	Used : 59.72 GB (27.01 %) Free : 161.36 GB (72.99 %)

Support :: Feedback

ManageEngine Free Tools

Figure 81. Display of Free Sql Performance Monitoring Tool (www.manageengine.com/sql-performance-monitor).

CHAPTER SEVEN
CONCLUSIONS AND RESULTS

In this study, importance of index was explained for query performance, and to understand index usage mentioned index structure. Knowing SQL Server Features and using them are important to improve query performance. Waiting query result can cause very huge problem when working with big data. So, this study's main aim was decrease the waiting time of query result.

To decrease the waiting time and improve query performance SQL Profiler was used to create trace file for track the queries. After save the trace file, this file used on Database Engine Tuning Advisor to analyze query needs, this analyze operation can be applied a query directly but, to analyze more than one query, trace file should be used. In this study, Database Engine Tuning Advisor analyzed the trace file, gave some index and statistics recommendation with their estimated improvement rate. Before apply the recommendations researches made on queries, after researches, recommendations applied and got better result. To measure the improvement, execution plans tooltips' estimated subtree cost values were used.

The other feature was Activity Monitor; with using this tool users can monitor CPU time, disk I/O and recent expensive queries. These are the main problems of performance so monitoring all of these with activity monitor panes is lifesaving because with using it the problem can be detected and find a solution for the performance. Recent expensive queries pane may give users the problematic queries so users can rewrite these queries or they can examine their execution plan to identify query necessity. The last SQL Server feature was execution plan this tool scrutinized. Users can detect query problem and find a solution by using tooltips.

After SQL Server general features, talked about relational algebra and its importance on query optimizer. Query tree and relational algebra expression were clarified.

The other important things were DMVs and DMFs, they do same thing with SQL Server tools. They can be used specifically on specific query. Therefore, they have some negative effects, when using DMV, users can be careful about date because this querying can bring all accumulative informations that are not useful for us. They may mislead the study.

Additionally, mentioned importance of statistics, using statistics and reading correctly them is important to understand the problematic queries and their solutions. Also, reading statistics is important for interpret the index’s effects on query. Therefore, this study mentioned all important point for performance improvement.

The last chapter was about related works, which are useful tools to use. Using different performance analyzer tools can help us to improve studies.

Consequently, different queries were used, some of them were complex some of them were simple. Different queries were chosen with different difficulty levels to could see different solutions and recommendations. Moreover, reached different solution for each queries, all solutions and recommendations analyzed and interpreted with using SQL Server information and applied their solutions to queries. The results were successful, the study reached its aim, and it improved queries performance on this study.

To improve this study new optimization tools and new MS SQL Server tools can be used. Query and index monitoring are mostly mentioned in this study to optimization and tuning, different database maintenance methods can be searched and applied to optimize and tune the database. These methods are not suitable for big database transections so, this study can be improved in this direction.

REFERENCES

Acungil, M. (2016). SQL Server'da DMV'ler (Dynamic Management Views) Üzerine. Mustafa Acungil; Eğitmen, Danışman, Teknik Yönetici: Http://Mustafaacungil.Blogspot.Com.Tr/2010/01/Sql-Serverda-Dmvler-Dynamic-Management.Html

Adar, İ. (2016). SQL Server 2016. Abaküs Yayınevi.

Agarwal, S. (2016, May 26). Activity Monitor In SQL Server. Http://Www.Sqlservercurry.Com/: Http://Www.Sqlservercurry.Com/2013/04/Activity-Monitor-In-Sql-Server.Html

Agarwal, S. (2016, 06 26). SQL Server Curry. Activity Monitor In SQL Server: Http://Www.Sqlservercurry.Com/2013/04/Activity-Monitor-In-Sql-Server.Html

Aktaş, İ. (2015). İsmail Aktaş. Database Performans Yönetimi: Http://Ismailaktas.Com.Tr/Ders-27-Database-Performans-Yonetimi/

Bolton, C., Langford, J., Berry, G., Payne, G., & Farley, R. (2014). Logicalread. SQL Server Query Optimization: Https://Logicalread.Com/Sql-Server-2012-Query-Optimization-W01/#.Wyiib4jyjiv

Corlăţan, C., Lazăr, M., Luca, V., & Petricică, O. (2014). Query Optimization Techniques In Microsoft SQL Server.

Dave, P. (2015, May 11). SQL Server-Introduction To Database Engine Tuniing Advisor(A.K.A DTA).

Davidson, L., & Ford, T. (2010). Performance Tuning With SQL Server Dynamic Management Views. Simple Talk Publishing.

Demircioğlu, M. (2012, 11 22). Execution Plan Index Davranışlarının İncelenmesi.

Fritchey, G. (2015). SQL Server Query Performance Tuning.

Fritchey, G. (2011). Query Performance Tuning: Start To Finish.

Galperin, E. (2011). Techfounder. Database Optimization Techniques You Can Actually Use: Http://Www.Techfounder.Net/2011/03/25/Database-Optimization-Techniques-You-Can-Actually-Use/

GHENCEA, A., & GIEGER, I. (2010). Database Optimizing Services.

Gözüdeli, Y. (2014). SQL Server 2014 & Veritabanı Programlama. Ankara: Seçkin Yayıncılık.

Host4ASP.Net. (2014). How To Optimize SQL Databases With Microsoft SQL Server Management Studio?

Https://Support.Smartbear.Com/Viewarticle/78858/. (2014).

Https://Support.Smartbear.Com/Viewarticle/78858/: Https://Support.Smartbear.Com/Viewarticle/78858/

Jones, D. (2014). Learn SQL Server Administration In A Month Of Lunch. Manning Publications.

Kline, K. (2010). Top 10 Tips For Optimizing SQL Server Performance. Quest Software.

Koch, R. (2016). Toptal. SQL Database Performance Tuning For Developers.

Kumari, N. (2012). SQL Server Query Optimization Techniques: Tips For Writting Efficient And Faster Queries.

Larsen, G. (2011, 06 03). Database Journal. Top 10 SQL Server Counters For Monitoring SQL Server Performance: Http://Www.Databasejournal.Com/Features/Mssql/Article.Php/3932406/Top-10-SQL-Server-Counters-For-Monitoring-SQL-Server-Performance.Htm

Measuring SQL Server Performance. (2017, 05 25). Measuring SQL Server Performance: Https://Support.Smartbear.Com/Viewarticle/78858/ Adresinden Alındı

Mehta, A. (2010). Performance Analysis Using Sql Server 2008 Activity Monitor.

Mehta, A. K. (2016). Performance Analysis Using SQL Server 2008 Activity Monitor Tool. Www.Mssqltips.Com: Https://Www.Mssqltips.Com/Sqlservertip/1917/Performance-Analysis-Using-Sql-Server-2008-Activity-Monitor-Tool/

Msdn.Microsoft.Com. (2014). Open Activity Monitor (SQL Server Management Studio): Https://Msdn.Microsoft.Com/En-Us/Library/Ms175518.Aspx

Msdn.Microsoft.Com. (2014). SQL Server Profiler: Https://Msdn.Microsoft.Com/En-Us/Library/Ms181091.Aspx

Mssqldude.Wordpress.Com. (2011, January 19). SQL Server 2008 R2 Performance Monitoring: Https://Mssqldude.Wordpress.Com/2011/01/19/Sql-Sevrer-2008-R2-Performance-Monitoring/

Petrovic, M. (2014a). Monitor SQL Server Queries-Find Poor Performers-Activity Monitor And Data Collection.

Petrovic, M. (2014b). SQL Server Activity Monitor.

Petrovic, M. (2014c, Feb). SQL Server Performance Monitoring With Data Collector-Part2-Set-Up And Usage.

Pinal, D. (2015). Sqlaquthority. SQL SERVER – Introduction To Database Engine Tuning Advisor (A.K.A. DTA): Https://Blog.Sqlauthority.Com/2015/05/11/Sql-Server-Introduction-To-Database-Engine-Tuning-Advisor-A-K-A-Dta/

Plecki, M. (2014). Technet Magazine. SQL Server: Https://Technet.Microsoft.Com/En-Us/Library/2007.11.Sqlquery.Aspx

Rusanu Consulting. (2014). How To Analyse SQL Server Performance: Http://Rusanu.Com/2014/02/24/How-To-Analyse-Sql-Server-Performance/

Sahtiyan, T. (2011a, Oct). Query Plan Görüntüleme Seçenekleri.

Sahtiyan, T. (2011b, Oct). SQL Server Profiler Ile Query Planları Toplamak.

Sahtiyan, T. (2011c, Feb). SQL Server'da Index Kavramı.

Shaw, G. (2009). Redgatehub. Finding The Causes Of Poor Performance In SQL Server: Https://Www.Red-Gate.Com/Simple-Talk/Sql/Performance/Finding-The-Causes-Of-Poor-Performance-In-Sql-Server,-Part-2/

Sahtiyan, T. (2011d, 08 24). Eksik Index'lerin (Missing Index) Belirlenip Oluşturulması Operasyonu.

Sahtiyan, T. (2011e). SQL Server'da İstatistik(Statistics) Kavramı.

Wort, S., Loforte, R., & Knight, B. (2017). Logicalread. Improving SQL Server Query Performance With Indexes: Https://Logicalread.Com/Improving-Sql-Server-Query-Performance-With-Indexes-W02/#.Wyikpijyjiv

Yeter, M. (2014, 02 24). MS How To. SQL Server Mimarisi – Nedir? Nasıldır?: Http://Www.Mshowto.Org/Sql-Server-Mimarisi-Nedir-Nasildir.Html

APPENDIX 1 – Dental Clinic Database SQL Code

Dental Clinic Database creating tables:

```
create table speciality(
SpecID int,
description varchar(30),
constraint PKS_Speciality primary key(SpecID)
)
create table Dentist(
DentistID varchar(15),
Name varchar(20),
Surname varchar(20),
SpecID int,
constraint PKD_dentist primary key(DentistID),
constraint FKS_Speciality foreign key(SpecID) references Speciality(SpecID)
on update cascade on delete no action
)
create table patient(
SSN bigint Not Null,
Name varchar(25) Not Null,
Surname varchar(25) Not Null,
Age int,
Gender char(1),
Phone nvarchar(15) Not Null,
DentistID varchar(15) Not Null,
Constraint PKP_Patient primary key(SSN),
Constraint FKD_Dentist foreign key(DentistId) references Dentist(DentistID)
on update cascade on delete no action,
constraint  Chk_gender check(charindex('F',Gender)>0 or charindex('M',Gender)>0 or
Gender is Null),
constraint Chk_Phone check(Phone is Null or
(Phone like '[0][0-8][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]') and
len(Phone)=11)
)
create table Appointment(
AppointID int Not Null,
date datetime,
description varchar(30),
DentistID varchar(15) Not Null,
SSn bigint Not Null,
constraint PKA_Appointment primary key(AppointID),
constraint FKAD_Dentist foreign key(DentistID) references Dentist(DentistID),
constraint FKP_Patient foreign key(SSN) references Patient(SSN)
on update cascade on delete no action
)
create table Medication(
MedID int Not Null,
Meddescription varchar(20),
constraint PKM_Medication primary key(MedID)
)
create table Prespcription(
PresID int Not Null,
MedID int Not Null,
SSN bigint Not Null,
DentistID varchar(15) Not Null,
Dosage varchar(15) Not Null,
Constraint PKP_Prescription primary key(PresID),
Constraint FKPD_Dentist foreign key(DentistID) references Dentist(DentistID),
Constraint FKPP_Patient foreign key(SSN) references patient(SSN),
Constraint FKM_Medication foreign key(MedID) references Medication(MedID)
on update cascade on delete no action
)
create table PaymentofTreatment(
TreatmentID int not Null,
Name varchar(30) Not Null,
Cost smallmoney Not Null,
Total_Amount as cost*(1+0.08),
constraint PKP_Payment primary key(TreatmentID)
)
create table Invoice(
invoiceNo int Not Null,
TreatmentID int Not Null,
SSN bigint Not Null,
DentistID varchar(15) Not Null,
constraint PKI_Invoice primary key(invoiceNo),
constraint FKT_Treatment foreign key(TreatmentID) references PaymentofTreatment,
constraint KFIP_Patient foreign key(SSN) references Patient(SSN),
constraint FKID_Dendist foreign key(DentistID) references Dentist(DEntistID)
on update cascade on delete no action
```

)

Procedures:

```
1. Alter proc[dbo].[Dental Clinic](@ID bigint)
    as
   Select i.SSN, sum(Total_Amount) as Payment
   From Invoice i, PaymentofTreatment p
   Where i.TreatmentID=p.TreatmentID and i.SSN=@ID
   Group by i.SSN
```

```
2. Alter Proc [dbo].[allowdentist](@username varchar (20))
      as
      Set nocount on;
      Select Distinct(p.SSN), p.Name, P.SurName, i.Name
      From patient p, Invoice r, Dentist d, PaymentofTreatment i
      where d.DentistID = r.DentistID and p.SSN = r.SSN and i.TreatmentID =
      r.TreatmentID
      and d.DentistID = @username
      Set nocount off;
```

Trigger:

```
Create trigger nochange
on Invoice
after update
```

```
as
if UPDATE(TreatmentID)
begin
if exists (
          Select *
              From Invoice r, deleted d
              Where r.SSN=d.SSN and d.TreatmentID<>r.TreatmentID)
              begin
               RAISERROR('Can Not Change Treatment',10,1)
               rollback
               end
```