Benchmarking Databases

"An investigation of the TPC-H benchmark suite and techniques used in the performance optimization of Decision Support Systems (DSS)."

Case study: Microsoft SQL Server 2008, 64 bit, Enterprise Edition

Samy Kabangu

Thesis submitted in partial fulfilment of the requirement of the degree of Bachelor of Science (Honours)

of Rhodes University

November 2009

ABSTRACT

Benchmarking a given database is a process of performing well defined tests on that particular database management system for the purpose of evaluating its performance. The query response time is one of the main criteria on which the performance of a database can be measured. An investigation of the TPC-H as the Transaction Processing Performance Council benchmark for decision support systems was performed during the execution of this project. Two techniques used in the performance optimization of decision support workloads were investigated, with Microsoft SQL Server 2008, 64 bit, Enterprise edition as the database management system .The first technique was the use of various kinds of index, including clustered, non-clustered and covering indexes. It was observed that queries run with indexes executed faster than queries run without indexes, but in this context a more important finding was that the embedded tool known as the tuning advisor gave effective recommendations about the indexes that should be created. This is valuable especially to help non-specialist users who are setting up a database to insert indexes on the correct columns.

The second technique was the investigation of the effect of query parallel processing on the execution time of queries. It was observed that queries run sequentially executed faster than queries run in parallel. Being counter-intuitive, this finding has drawn comment from Microsoft and others in the database user community, indicating a recognition of the problem. The current belief is that the parallel execution times are explained by resource contention, but this needs further investigation before applying the use of query parallel execution on SQL Server 2008 in a production environment.

ACKNOWLEDGEMENTS

I would like to thank my supervisor John Ebden for his guidance, support and more especially for his patience with me throughout the completion of this thesis. I acknowledge the financial aid of the ANDREW MELLON PRESTIGIOUS SCHOLARSHIP for giving me the opportunity to further my studies at Rhodes University. I acknowledge the financial support from Telkom SA, Business Connexion, Converse SA, Verso Technologies and THRIP through the Telkom Centre of Excellence at Rhodes University. Last but not least I would like to thank my parents Louis and Julienne Kabangu for their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

Chapter 1- Introduction

1.1 Statement of the problem

As most applications in the area of computer science and information technology industry seek for high performance, the database computing arena is not left behind. New database products are being developed with much concern about the speed of loading, modifying and retrieving stored data from the database. The Transaction Processing Performance Council (TPC) provides various tests meant to evaluate the performance of database management system products under specific workload, operating systems and hardware platforms.

This research aims to an investigation of the TPC-H benchmark suite as well as the experimentation of techniques used in the performance optimization of decision support workloads. As a case study, Microsoft SQL Server 2008 is used as the database management system product on which to run the TPC-H benchmark tests.

1.2 Project motivation

With the advent of Business intelligence among many other applications of decision support systems requiring the storage of large amount of data into databases for future analysis, querying such systems might take hours or even day of execution runs. In some cases, the retrieval of data might be required to be performed on a daily basis. With such constraints, It becomes evident about not only to use databases that scale and perform well but also finding different techniques of optimizing the performance of such systems with respect to the decision support workload being run on them. Through Benchmarking and scalability testing, Database professionals can simulate such applications before deploying them in a production environment.

1.3 Project overview

The rest of this thesis comprises five chapters. Chapter 2, provides an in-depth background of database benchmarking techniques, a description of the TPC-H as the current decision support benchmark suite for implemented by the Transaction Processing Performance Council (TPC) as well some techniques used in database performance optimization. Chapter 3 provides the design considerations under which this project was implemented. Chapter 4 is related to the methodology used to tackle the problem posed in project. Chapter 5 provides the results gathered during the experimentation phase of this project as well as their interpretations. Chapter 6 provides some recommendations, future extensions and conclusion of this project.

Chapter 2- Related Work

This chapter provides:

- Some fundamental concepts in database benchmarking tests,
- Some background information about the TPC-H benchmark suite,
- Some techniques used in database performance optimization.

2.1 Introduction

Benchmarking a database is the process of performing well defined tests on that particular database for the purpose of evaluating its performance [24]. The Response time and the throughput are the two main criteria on which the performance of a database can be measured [17]. Specific parameters and settings external as well as internal to the database management system need to be taken into consideration. These parameters include the hardware used to test the system, the internal configuration of the database engine, the operating system configuration as well as the database design and implementation [2][22][34]. All the parameters mentioned play an important role in the overall performance of a database management system (DBMS).

2.2 Significance of database benchmark tests

Benchmark test results facilitate means for cross platform comparisons of various database management systems by providing valuable information to database professionals on whether to utilize a particular database product. Within an organization, the workload supported by a database system might increase as the business expands; Proactive benchmark scalability testing can be beneficial in preventing bottlenecks [23]. Furthermore, there might be a need of migrating from one hardware platform, or system software or database product to a newer version or release. Database benchmarks tests can be valuable by providing a proof of concept that facilitate the job of the DBA to make an apple to apple comparison between different software releases [21][23]. Finally, Benchmarks tests promote innovation due to competition between hardware manufacturers, operating system developers and database vendors [34].

2.3 Benchmarking Process flow chart

Benchmarking is a difficult and never ending process that requires a lot of patience and discipline [9]. As the benchmarking process is being executed, measurements about the overall performance of the system have to be collected as various key configuration parameters specific to the hardware; operating system and database management system are being altered if necessary in order to improve the performance of the system under test [9]. We would like to mention that the TPC benchmarks are used to evaluate the performance without any attempt to modify the configuration parameters of the system under test [7][30]. Any benchmark test performed using the TPC benchmark suites with the purpose of improving the performance of the system under test is qualified as “special” [30].

The flow chart below illustrates the steps involved in the benchmarking process [6]:

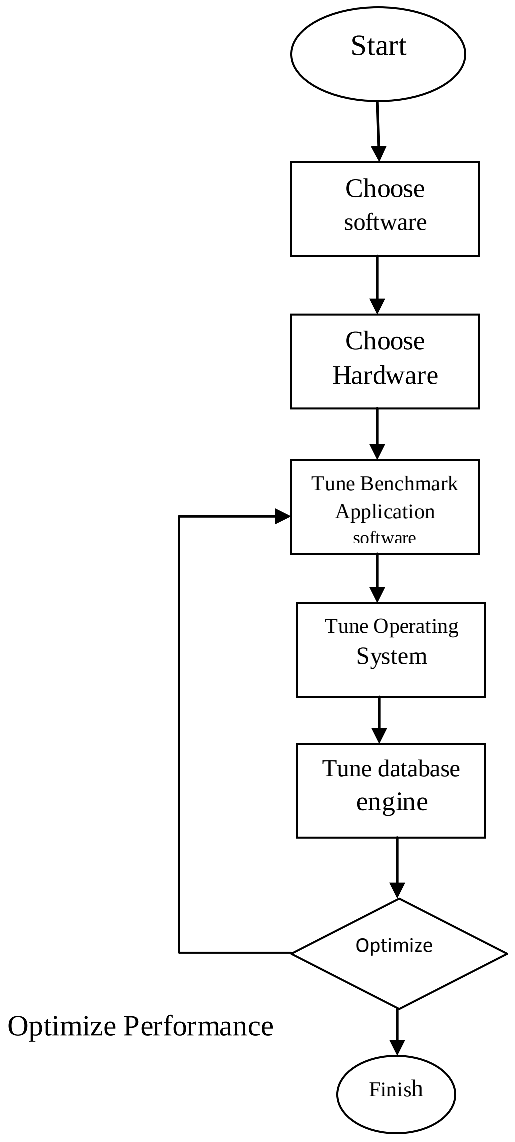


Figure 1 : Benchmarking Process Flow chart [9]

2.4 Types of database Benchmarks

Benchmarking processes can be classified into three types: Industry-standard benchmarks, vendor and customer-application benchmarks [9].

2.4.1 Customer-application benchmarks

A customer-application benchmark is the benchmark performed within an organisation. They have the advantage of meeting the specific requirement of the organisation in terms of workload and hardware implementation but they are costly and time consuming endeavours usually difficult to undertake [34].

2.4.2 Industry standard benchmarks

Industry standard database benchmarks were developed to provide a cross platform comparison among various database products in terms of performance and prices. The published performance test results are measured depending on specific database workload types.

The most common workload types are:

- Online Transaction Processing (OLTP),
- Online Analytical Processing (OLAP); eg: An online business intelligence system workload against which users submit queries to answer complex business questions [25].
- Decision support system

The decision support workload type will be of particular interest in this literature review since it is implemented under the Transaction Processing Performance Council (TPC) benchmark suite "TPC-H". It is also the only TPC benchmark suite that has no clients and no network components [34]. So the TPC-H benchmark suite appears to be the most suitable one for this research due to the limited resources that we possess at hand.

2.4.3 Reservations about industry standard benchmarks

Though they provide means of knowing the performance of database products, their adoption might be somewhat difficult in that:

- They simulate real world workload which might not reflect the actual workload of a particular application of interest [13][21].
- They are performed on specific hardware and operating system which makes their duplication not feasible especially for custom applications that do not match the platform requirements on which the benchmark tests were run.
- It is also difficult to compare database systems run on different hardware platforms because of the different machine architecture under which they are manufactured [21].
- Database vendors use techniques such as preloading data and the SQL execution plans into memory (RAM) in order to avoid disk I/O access overhead so as to improve the benchmark performance [3].

2.4.4 Transaction Processing Performance Council (TPC)

The TPC is a non–profit corporation which defines transaction processing and database benchmarks by publishing to the industry verifiable TPC performance data [29]. The term transaction viewed from the business perspective is regarded by the TPC as a commercial exchange of goods, services or money. As a computer function, “transaction” refers to a set of operations comprising disk read/writes, operating system calls, or some data transfer from one subsystem to another [29].

The TPC provides different benchmark suites designed according to specific workload type and applications requirements. The TPC benchmark suites currently valid are given in the table below [29]:

TPC Benchmarks	Workload and applications Types
TPC-App	• An application server and web services benchmark • Focuses on the performance capabilities of application server systems
TPC-C	• Simulates an application where a population of users executes transactions against a database (OLTP)
TPC-E	• The new On-Line Transaction Processing • Is scalable, the workload generated can be varied to represent the workload of different-size businesses
TPC-H	• Decision Support benchmark consisting of ad hoc queries and concurrent data modifications.

Table 1-TPC Benchmark suites

2.4.4.1 TPC-H

The TPC Benchmark™H (TPC-H) is a decision support benchmark consisting of a suite of business oriented queries and concurrent data modifications [30]. The queries and the data populating the database have been selected to have a broad industry-wide relevance [30].

The TPC-H Benchmarks simulates decision support systems that [30]:

- Examine large amount of data;
- Execute queries with a certain degree of complexity;
- Give answers to critical business questions (operation).

Due to the ad hoc nature of the TPC-H queries, their execution time can be very long. That makes it difficult for the database administrator to optimize the database system as opposed to applications such as OLTP where the nature of queries as well as that of the workload is known in advance [10]. The properties of the TPC-H as reported by the TPC Benchmark™H standard specification Revision 2.8.0 are given as follows [30]:

- Give answers to real-world business questions;
- Simulate generated ad-hoc queries
- Are far more complex than most OLTP transactions;
- Include a rich variety of operators and selectivity constraints;
- Are executed against a database complying with specific population and scaling requirements;
- Generate intensive activity on the part of the database server component of the system under test;
- Are implemented with constraints derived from staying closely synchronized with an on-line production database.

2.4.4.2 TPC-H Database design and implementation

The TPC-H benchmark defines different sizes of the database according to specific scale factor as follows: 1GB, 10GB, 30GB, 100GB, 300GB, 1000GB, 3000GB, 10000GB, 30000GB. The scale factor of 1(1GB) is the minimum required size for a test database. Any database size not mentioned is not permitted by the TPC. This requirement is meant to encourage comparability of the results and to ensure a significant actual difference in test database sizes [30]. The TPC-H Benchmark models the analysis end of the business environment where trends are computed to support decision making of sound business decisions [30].

The TPC-H benchmark database has been designed to be in the third normal form [10]. That is, it has the following properties [4]:

- All the key attributes are defined;
- There are no repeating group in the tables;
- All the attributes are dependent on the primary key;
- No attribute is dependent on only a portion of the primary key;
- It contains no transitive dependencies

The entity relationship model is given by the schema below:

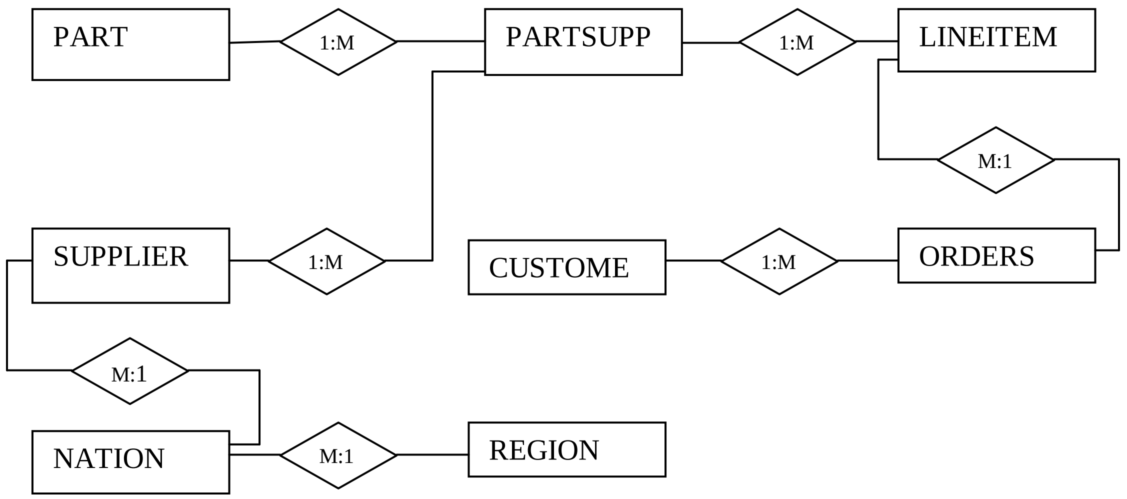


Figure 2- TPC-H Benchmark database schema

The database consisting of eight tables is populated using a designed program that comes with the TPC-H benchmark suite called DBGEN. The maximum cardinality supported by each table is dependent on the scale factor used. The tables “SUPPLIER” and “LINEITEM” contain about 83% of the total data stored in all the tables [10].

It is relevant for us to mention that according to [24], “TPC benchmark kits for most state-of-the-art database systems are not readily available” and writing and tuning the benchmarks to meet the required specifications of the workload of interest on a given database system may require over six months of experimentation even by trained database system managers.

2.4.4.3 TPC-H Workload

The TPC-H benchmark workload consists of a database load, the execution of twenty-two read only queries running on single as well as in multiple users mode and two batch update statements (RF1 and RF2) [10] [24]. RF1 inserts new rows into the tables LINEITEM and ORDERS while RF2 removes the same number of rows from those tables [10][24].

The Database load consists of constructing the test database which includes the process of loading the data into the database , the creation of tables, indexes, the definition and validation of constraints, the collection of statistics and the configuration of the system so that it can meet the ACID requirements. Synchronizing the loaded data on RAID (Redundant array of independent disks) devices may also be taken into consideration.

The program used in the TPC-H benchmark to generate queries against the test database is the QGEN program. It is written in ANSI'C' and has been ported to a large number of platforms [11]. Minor syntactic modifications of the TPC-H queries are permissible so that they can be run on specific commercial database application and all the tables created during the execution of a query must meet the ACID proprieties [30].

Each and every TPC-H benchmark query is defined by the following components [30]:

- The business question, which illustrates the business context in which the query could be used;
- The functional query definition, which defines, using the SQL-92 language, the function to be performed by the query;
- The substitution parameters, which describe how to generate the values needed to complete the query syntax;
- The query validation, which describes how to validate the query against the current database.

According to Zaharioudakis et al, modern decision-support queries involve joins, arithmetic operations, complex aggregations and nested sub-queries which make them complex [35]. The ad hoc nature of the TPC-H benchmark queries combined with their complexity make their execution time longer. They may take hours or even days of execution runs.

Graefe et al., define a complex query as the one requiring a number of query processing algorithms to work together and a large database, as the one using files with sizes ranging from several megabytes to many terabytes [11].

Some works have been done with the aim of optimizing decision support database applications that are mainly characterised by their database size being large. Much detail will be given in the performance optimization section that follows.

2.5 Database Performance optimization

In order to obtain an optimum performance from a database system, the design and implementation model of the database must be well performed [4]. That alone does not guarantee a complete solution to the performance issues. Other techniques such as query optimization need to be taken into consideration as well. Key configuration variables specific to a particular database product affecting the query optimization process might need to be altered.

2.5.1 Database engine

Graefe et al., define query processing as "the component filling the gap between the database query languages and the data storage systems in a database management system" [11]. It comprises the query optimizer that translates queries written in a high level query language into a series of operations that are implemented in the query execution engine. The query optimizer has the responsibility of finding a query evaluation plan that minimizes the performance cost measures of a database [11]. These performance measures include:

- The database user's wait for the first or last result item;
- CPU cycles;
- Memory costs(as maximum allocation or as time-space product);
- I/O (Input/Output) data transfer;
- Network time and effort;
- Total resource usage

2.5.2 Query execution plan

A query plan as stated by Shao et al. is made of “a cooperating tree of operators”. Since a logical operator may be executed using a combination of various physical operators, a single query can be executed using one of the many possible query execution plans that can be produced by the query optimizer.[25].

The difference between the logical operators and physical operators as stated by Graefe et al., resides in the fact that physical operators implement logical operators [11]. A logical operator defines how the query can be expressed in the data model where as a physical operator is more specific to the query processing system.

The difference is illustrated by the figure 2.3 below:

Logical operator Physical operators

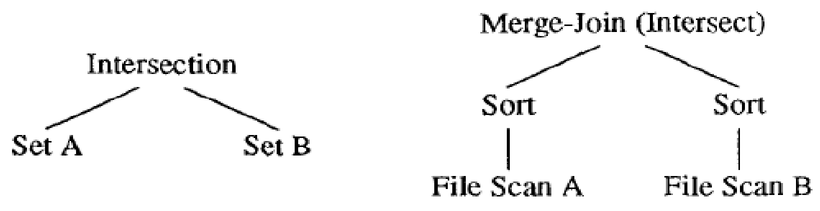


Figure 3- Logical and physical query operators [11]

The most used physical operators as described by Shao et al. are [25]:

- A table scan: Reads through an entire table and generates a stream of records that satisfy the condition (predicate) part of the query statement; An index scan operator provides the same results as the table scan by using an index to access only records that meet the predicate;
- Table joins: Match rows from two tables based on an equality or other condition on common fields. Joins may be implemented using the nested loop, sort-merge or hash join algorithms.

- Order-by clauses are implemented using the sort operator that sorts records in the input table with respect to a subset of fields.
- Group-by: categorises the input records into groups based on a subset of fields and outputs of the groups. It can be implemented using sorting or hashing.
- Aggregate: Refers to a function such as sum, max, min, count, avg etc. that can be applied on the input records to output a single value.

2.5.3 Indexes

The database tables of decision support systems are “heavily indexed” and the raw data are structured in such a way that they can support all the various types of queries that are being used [16]. Having the knowledge that indices can improve greatly the query execution time if they are properly designed to suit the workload of interest; some care must be taken when creating them. Nevertheless, Microsoft SQL Server provides a means of helping with the analysis of the database environment and in the selection of appropriate indexes via the Database Engine Tuning Advisor tool [27].

The three main types of indexes considered in analysing queries in SQL Server are [5]:

- Clustered indexes: which index directly to the column on which is created and order the data in that column in a given manner. A primary key is a good candidate for a clustered index.
- Non-clustered indexes: which index one or many columns on which is created through the primary key (clustered index).Good for non-clustered indexes are, foreign keys, columns involves in join, Group by, and Having operators.
- Covering indexes: are non clustered indexes in which extra columns have been included. They have the advantage of covering all the columns involved in a query, hence the actual table need not to be accessed since the index table could retrieve directly the records specified in that particular query [32].

As an illustration, using indexes could result in great performance improvement when running them using the TPC-H query 1 given below:

```
select

        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
        sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
        l_shipdate <= DATEDIFF (DD, 3, '1998-12-01')
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
```

Without indexes, the entire LINEITEM table consisting of 6000000 rows in a database of size 1GB has to be scanned so that the records meeting the "where clause" can be retrieved. The result of this query consists of only four records and it is probable that it might take quite some time to execute. So it is clearly evident that the use of indexes can dramatically reduce the execution time of such queries.

2.5.4 Parallel query execution

As opposed to Online Transaction Processing workload, the decision support workload which does not require frequent insertion, deletion or update operations can benefit significantly from parallel query execution as they are characterized by large tables, and complex queries involving multiple join operators between tables.

The three forms of parallelism in query execution that are common to designers of database query processing systems are given as follows:

- Inter-query parallelism
- Inter-operator parallelism
- Intra-operator parallelism

Most of today's database management systems make use of the inter-query parallelism and specialized version of the inter-operator parallelism called "horizontal inter-operator parallelism" [12].

2.5.4.2 Inter-query parallelism

Inter-query parallelism refers to the ability of a given database management system to execute multiple queries concurrently [12]. The drawback of this technique is that it exhibits resource contention as different queries would require accessing the same object.

2.5.4.3 Inter-operator parallelism

Inter-operator parallelism involves a parallel execution of different operators such as selection, join or intersection in single query [12]. It is also referred to as pipeline parallelism. The two ways in which inter-operator parallelism can be used are:

- Vertical inter-operator parallelism which involves the execution of query operators composed by a "producer and consumer relationship" in a pipeline such that tuples output by the producer are being received by the consumer as they are being produced.
- Horizontal inter-operator parallelism which involves the execution of independent subtrees in a complex bushy-query evaluation plan concurrently. It is also referred to as bushy parallelism.

2.5.4.4 Intra-operator parallelism

Intraoperator parallelism is a form of query processing parallelism which provides a means of executing a single operator in a query plan in multiple processes [12]. It involves running more than one instance of a single operator on different processors (machines) concurrently.

2.5.5 Query parallel processing on SQL Server

SQL Server is capable of executing query parallel processing using multiple processors simultaneously when is running on a multiprocessor hardware platform [26]. It is only the task of the query optimizer to decide depending on the cost whether a query has to be executed in parallel or sequentially [26]. More often, a complex and expensive query processing many rows will likely be a candidate of a parallel plan than a query that only processes few rows. The key configuration variables that need particular attention are the “cost threshold for parallelism” and the “max degree of parallelism (MAXDOP)” both variables can be configured using the sp_configure stored procedure. If the query cost exceeds the value defines by the cost threshold for parallelism, the query optimizer will attempt to generate a parallel plan for that particular query. The max degree of parallelism defines the number of cores (CPU) that should be used during the execution of a parallel query [26].

2.5.5.1 Exchange Operators

Query exchange operators are use in a SQL Server execution plan to prepare a given query for parallelism. They provides in SQL Server execution plan:

- Process management,
- Data redistribution and
- Flow control.

The three type of exchange operators are: Distribute streams, repartition streams and gather stream.

2.5.5.1.1 Distribute streams

It takes a single input stream of records and output multiple streams. So each record from the input stream appears in each of the output streams [26].

2.5.5.1.2 Repartition streams

It takes multiple input streams of records and output multiple streams of records as well. Each record of the input streams is placed into one output stream. [26]

2.5.5.1.3 Gather streams

It takes many input streams of records and output a single stream of records. If the Gather streams operator is order preserving, then all the input streams should be ordered [26].

2.5.6 Workload optimization

After running the TPC-H benchmarks on IBM DB2 UDB V.7.2 using a 4 way 733MHz Intel Pentium III server, Shao et al., observed that most of the TPC-H queries execute basic query operations such as sequential scan or join [25].

The query optimizer's suggested plans for the TPC-H queries were found as follows [25]:

- 50% of the queries were dominated by tables scan(Over 95% of their execution time is estimated to be due to table scans);
- 25% of the queries spent more than 95% of the time executing nested-loop joins;
- 25% of the remaining queries executed table scans for about 75% of the time on average and nested-loop joins for about 25% of the time on average

The obtained results were said to be counterintuitive considering the complexity and depth of a TPC-H query plan [25]. Such results could be explained by the filtering being done at the lowest levels of the operator tree and the size of the result being reduced as the execution continues to the upper levels of the tree. Shao et al. concluded by suggesting the scaling down of the TPC-H workload by constructing representative queries that execute the dominant operators (Sort and join) and the use of small datasets that fit in the research test bed [25]. The primary characteristics of interest for measuring performance of the system under test were [25]:

- Query execution time breakdown;
- Memory stall time breakdown in terms of cycles lost at various cache levels and TLBs;
- Data and instruction cache misses per instruction at each level branch;
- Branch misprediction per instruction

Shao et al. noted that on average the processor remained idle more than 80% of the time when executing the TPC-H queries [25].

Wasserman et al., after analysing the TPC-H queries run on DB2 UDB, grouped them into 4 classes based on their processing time, I/O and n-way table joins characteristics as described by the table below [33]:

Classes	Query Number	Characteristics
Class 1	Q11, Q14, Q5, Q12, Q8, Q7, Q1, Q3, Q4, Q10	• Medium-complexity Query • High Response times • Moderate CPU and I/O usage
Class 2	Q2, Q20, Q17 (Q19 and Q6 are borderline)	• Simple queries which are I/O-bound and join small number of tables
Class 3	Q9, Q18, Q21	• Large and complex queries which are long-running • Have large number of tables joined • Exhibit high sequential and random I/O usage
Class 4	Q13, Q22, Q15, Q16	• Trivial queries • Short run times • Small number of tables joined • Exhibit high CPU utilization

Table 2-TPCH-Queries classification

2.5.7 Performance Bottlenecks

It is of great importance to pinpoint performance bottlenecks before embarking on the process of optimizing the system under test. The system monitor embedded on SQL Server or on the operating system (Microsoft Windows Server 2008) may be useful in monitoring and analyzing the system performance behaviour.

The performance data being generated by a particular component of the system under test is represented by counters that reflect the status of that component. An example could be percentage (%) processor time buffer cache hit ratio [6].

2.5.7.1 Bottlenecks identification and resolution

Dam et al. suggested two ways of resolving bottlenecks that are: Increasing the resource throughput and/or decreasing the arrival rate of requests at the resource [6]. The first one can be accomplished by adding more resources such as memory, disks, processors or network adapters. The second one may be achieved by adding appropriate indexes on a table to limit the amount of data that can be accessed. Decreasing the arrival rate of data refers to the process of identifying the causes of high I/O requests to the disk subsystem with the aim of minimizing their number [6].

2.5.7.2 Memory Bottlenecks

A memory bottleneck will rapidly impact on other resources such the processor or the disk [6]. SQL Server reserves memory for data as well as for the query execution plan using the memory pool which consists of a collection of 8KB buffers to manage data pages, plain cache pages and free pages. SQL Server manages its memory by dynamically growing or shrinking its memory pool size dynamically so as to maintain free physical memory between 4MB and 10MB [6]. The dynamic memory configuration is recommended for SQL Server with “min server memory” set to zero and “max server memory” set to the maximum physical memory of the system. If other applications are being run on the server while SQL Server is also running then the “min server memory” value should be configured to 50% of the max server memory value in order to prevent those applications from using memory that SQL Server might need [6]. The “min server memory” and “max server memory” values can be configured using the “sp_configure” system stored procedure.

Memory bottlenecks can be resolved by [6]:

- Optimizing the application workload (especially queries)
- Allocating more memory to SQL Server
- Increasing the system memory
- Using extended memory within SQL Server

2.5.7.3 Disk Bottlenecks

Intensive disk operations on the resources residing on the disk can result in performance decrease of SQL Server since it usually uses much of the hard disk space [6]. Disk counters can be used to monitor disk performance. The list below give an overview of some of those counters [6]:

- “% Disk Time”: Monitors the disk read/write activities and provides the result in terms of percentage which should not be continuously high. If the “% Disk Time” counter is consistently having a value of more than 85% then upgrading the disk subsystem could be one of the options to bring that value down. A more suitable solution would be the one of avoiding going to the data disk frequently that is caching the disk contents in memory (Buffer cache).
- Current Disk Queue Length: Provides the number of requests outstanding on the disk subsystem at the time the performance data is collected. The current disk queue length is used to support the results provided by the “% Disk Time Counter”.

Disk bottlenecks can be resolved by [6]:

- Optimizing the workload;
- Using a faster disk drive;
- Creating multiple files and file groups
- Placing the table and indexes for that table on different disks
- Saving the log file to a separate physical drive
- Using a RAID array

2.5.7.4 Processor bottlenecks

The table below gives the normal values that can be provided by the processor objects counters [6]:

Object(Instanace[InstanceN])	Counter	Description	Value
Processor(_Total)	% Processor Time	Percentage of time the processor was busy	Average value<80%
	% Privileged Time	Percentage of processor time spent in privileged mode	Average value<10%
System	Processor Queue Length	Number of requests outstanding on the processor	Average value<2
	Context Switches/sec	Rate at which the processor is switched from one thread to another	Average value<1000 per processor

Table 3-Processor performance bottlenecks

Any value greater than the ones specified in the table 2.3 above, might reveal some performance bottlenecks on the part of the processor.

2.5.7.5 Overall performance on SQL Server

It might be useful to examine some general aspects of SQL Server itself besides the hardware resource utilization. The table below lists counters for SQL Server overall performance [6]:

Object(Instance[InstanceN])	Counter
SQLServer:Access Methods	FreeSpace Scans/sec, Full Scan/sec
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec, Lock Wait Time (ms), Number of Deadlocks/sec
SQLServer:SQL Statistics	Batch Requests/sec, SQL Re-Compilation/sec
SQLServer:General Statistics	User Connections

Table 4 - SQL Server overall performance counters

The counters that might be relevant to us are explained below:

- FreeSpace Scan/sec and FullScan/sec: Both are counters provided by the access methods object which monitor how the logical pages within the database are accessed. Monitoring the methods used to access database pages can help in improving the query performance by adding or modifying indexes or by rewriting the query [13]. The FreeSpace Scan/sec provides the number of scans per second that were initiated to search for free space within the pages already allocated to an allocation unit to insert or modify record fragment. Each scan may find multiple pages [13]. The full Scans/sec provides number of unrestricted full scans per second which can be either base-table or full-index scans [13].
- Total Latch Wait Time: Latches are used to protect the integrity of the internal structures of SQL Server such as table row. They are not directly controlled by the users. The Total Latch Wait Time monitors total latch wait time for latch requests that had to wait in the last second [6].
- Lock Timeouts/sec and Lock wait Time: The Lock Time out should be expected to be zero and the Lock Wait Time to be low otherwise a blocking might be occurring in the database. Identifying the costly queries using the SQL Profiler tool provided by SQL Server might be one of the options to resolve the blocking problem [6].

- Number of Deadlocks/sec: This counter should be expected to be zero; otherwise the problematic request has to be identified [6].
- SQL Re-Compilation/sec: The reusability of an execution query plan is beneficial for the fact that generating an execution plan for a stored procedure query requires more CPU cycles [6]. So the SQL Re-compilation/sec counter is used to analyze the non-reusability of a stored procedure. A value of zero should consistently be reflected by the SQL Re-Compilation/sec counter. If a nonzero value is consistently reflected, that means there is an overhead on the side of the processor because of the recompilation of the stored procedure. The SQL Server Profiler should be used for further investigation [6].

2.6 - Chapter Summary

This chapter aimed to provide some background concepts about the TPC-H as the Transaction Processing Performance Council (TPC) benchmark suite for decision support systems as well as some techniques used in database benchmarking and performance optimization. It also provides concepts relevant to the understanding of the experimentations that will performed in this project as well as the interpretations of the results that will be obtained.

The next chapter provides a description of the design considerations relevant to this project.

Chapter 3-Design considerations

The aim of this chapter is to provide a description of all the components making part of the system specification considered for this project. This includes the hardware platform, the operating system, the database management system product and other software tool used. A brief explanation of the motives behind their choice is also provided. Finally, this chapter also includes a description of the structure of database used as well some specific configurations variable of interest.

3.1 System specification

The system under test has the following features:

- Microsoft Windows Server 2008 Enterprise Service Pack 1, 64 bit version as the server operating system.
- As database management system products, Microsoft SQL Server 2008, 64 bit version

Both the operating system and the database management system run on a Hardware platform having the following features:

- 64 bit machine (Proline)
- Intel® Core ™ 2 Quad CPU @ 2.66GHZ (4 CPUs)
- 4 GB of RAM
- 500GB of Hard Disk

The motivations driving the choices of the database product and operating system are given in the following sections.

3.2 Microsoft Windows Server 2008

As the latest server operating system released by Microsoft and the recommended one for Microsoft SQL Server 2008 database, Microsoft Windows Server 2008 is the subject of positive customer feedback such as cost saving as well as resource optimization which are essential in today's competitive IT industry [18]. Windows Server 2008 is able to achieve this due to the features that come built into it. Systems such as the Microsoft hypervisor-based server virtualization technology "Hyper-v" enables Windows Server to take advantage of the multiprocessor architectural technology being offered by current hardware platforms [18]. Processor utilization level of servers running Windows Server 2008 can be monitored using a balanced power policy which provides a means of dynamically adjusting the processor performance level according to the workload, hence limiting the server processors power consumption as well as the cooling cost [18]. These are among the reasons that motivated the choice of Windows Server 2008 as the server operating system of this project.

3.3 Microsoft SQL Server 2008

SQL Server 2008 is believed to be the most robust and comprehensive database product released by Microsoft to date. Its combination with Windows Server 2008 is said to be a powerful platform for running mission-critical data and business intelligence solutions with high security, compliance and availability. Being more scalable, SQL Server 2008 is said to be meeting the needs of data warehouse of the largest enterprises with ease. These are among the characteristics that set it apart from other database products on the market [17]. SQL Server 2008 also provides tools for collecting and analyzing performance data such as the "Tuning Advisor" for performance optimization. Experimenting with the features provided by SQL Server 2008 in order to attain the objectives assigned in this project as well as verifying the claims made in its favour were the principal motives behind the choice SQL Server 2008 as the database product for this project [17].

3.4 Benchmarking software tools

Since the TPC does not currently provide a readily available benchmark kit for SQL Server, a third party benchmarking software tool “Benchmark Factory for databases version 6” was used to generate the TPC–H database workload. Another third party performance monitor tool “Spotlight for SQL Server Enterprise” was used to support the performance monitors embedded in Microsoft Windows Server 2008 and Microsoft SQL Server 2008. Spotlight for SQL Server Enterprise provides a means of visualizing the main performance counters measuring the overall performance of the system under test such as “% processor time”, buffer hit ratio and the average disk queue length.

3.5 SQL Server Performance monitors tools

Microsoft SQL Server 2008 provides performance monitors tools that are valuable in collecting and analyzing the database performance results as well as in identifying the causes and sources of potential system bottlenecks with suggestions for resolving them. Below are some of the tools that we found useful for this projec**t.**

3.5.1 Microsoft SQL Tuning Advisor

The Database tuning advisor automatically suggests optimal set of indexes, indexed views, partitions and table statistics based on the type of workload being analyzed and the physical implementation of the database [27]. The tuning advisor can also suggest the modification of the physical design structure of the database. For this project we were only concerned with the optimal indexes suggested since the database schema is part of the TPC-H specification. During the process of generating the database workload using the benchmark factory for databases benchmarking software, indexes were also created as recommended by the TPC-H specifications. Nevertheless we still used the SQL Server database tuning advisor indexes suggestions in order to experiment the capabilities of this tool especially for decision support workload such as the TPC-H workload since some significant performance improvement in terms of query response time was observed when indexes scripts suggested that by the tuning advisor tool were used. Further details will be given in the discussion and evaluation chapter of this project.

3.5.2 Microsoft SQL Server profiler

The SQL Server Profiler is a tool that provides a graphical user interface for capturing and saving SQL Server events to a file or table for later analysis [31]. These saved files that can also be referred to as SQL Trace can be used for the purpose of monitoring SQL statements and stored procedure that are affecting the system performance by running slow, indexes and other similar experiments.

3.6 TPC-H Database

Designed to be in the third normal form, any TPC-H database consists of eight tables whose cardinalities depend on the scale factor being used except for the NATION and REGION tables. The scale factor determines the size of the database generated for a given test. Much detail about the schema and size of different database is provided in Chapter 2- Related work. The database tables have the following cardinalities:

	Cardinality (Number of rows for each table)		
Tables	*Scale Factor 1 (1GB)*	*Scale Factor 10 (10GB)*	*Scale Factor 30 (30GB)*
PART	200000	2000000	6000000
PARTSUPP	800000	800000	24000000
LINEITEM	6000000	60000000	1800000000
SUPPLIER	10000	100000	300000
CUSTOMER	150000	1500000	4500000
ORDERS	1500000	15000000	45000000
NATION	25	25	25
REGION	5	5	5

Table 5 -Table's cardinalities

3.7 Key configurations variables of interest

As stated by Microsoft, Microsoft SQL Server setting variables are configured for optimality. However such recommendations are not cast in stone since experience has proven that altering some of these internal settings can result in some performance gain. Driven by an experimental mind, we have assigned ourselves the task of modifying some of these variables that can impact on the performance of query execution. These variables of interest are given as follows:

- Maximum degree of parallelism
- Cost threshold of parallelism
- Max worker threads

Detailed explanations of these variables will be given in chapter4- Methodology.

3.8 Chapter summary

This chapter gave a description of the components making part of the system specification that are the hardware platform and software chosen as well the reason behind their choice. A brief description of the database structure, the table's cardinalities and specific configuration settings of interest were also reviewed. The next Chapter provides more detailed information about the methodology used to solve the stated problem.

Chapter 4-Methodology

The approach used in solving the problem posed in this project was to carry out an investigation on the TPC-H queries workload before running the benchmark test on our custom application being configured with its default settings. Performance monitors were then used to collect information about the database performance in both real time and after running the queries. Based on the investigations done on the TPC-H workload as well as the results measured by the performance monitors, the performance optimization of the system under test was guided through two main operations given as follows:

- The use of the tuning advisor suggested indexes,
- The experimentation of different configuration variables pertaining to parallel query processing.

4.1 Database loading process

Three different databases named “TPCH_1GB”, TPCH_10GB1”, TPCH_30GB” were respectively generated with the TPC-H workloads of scale factor 1, 10, and 30. As required by the TPC-H specification, the three different scale factors were selected in order to observe significant differences in query response between these three different scale factors. The process of generating the workload was made easy through the use of the “Benchmark factory of databases” software tool which loaded the required data for the three different sizes of databases.

4.2. Benchmark tests - with default configuration

After the process of loading the data into the three databases, each of the twenty-two TPC-H queries was then run against each of the three databases set to their default settings. Performance monitor tools were used to capture the health of the system under test. Performance test results such as the query response time, CPU average spent for each query as well as the number of disk reads occurred during the execution of each query were recorded for later comparison with the results obtained after the configuration of the three databases. More detailed explanation is provided in chapter 5-Result of this project.

4.3 Benchmark tests – Performance optimization

The performance optimization process was guided by the performance test results collected with all the databases set to their default configurations as well as the results about the health of the system under test provided by the performance monitors. The two methods of optimizing the system under test were: the use of indexes and the experimentation of the settings pertaining to the execution of queries in parallel. Their selection was based on the type of performance bottleneck occurring on the server. A high number of disk activities was reported by the performance monitors. This selection was also supported by the literature survey provided in Chapter 2-“Related work” in section 2.5.3 and section 2.5.4 of this project concerning the optimization of decision support systems workload.

4.3.1 The use of indexes

After the process of recording the query performance results with each of the three databases set to their default configurations, a set of seven queries with the highest response time in both, the database generated with a scale factor 10 (10 GB) and the one generated with a scale factor of 30 (30 GB) were analyzed by the tuning advisor tool. The tuning advisor then generated non clustered index scripts which were run against each of the two different databases. The selected seven queries were again run against each of the two databases (scale factor =10 and 30) and their execution time was compared with the ones run when no indexes were created on both databases except the indexes that were created during the process of generating the workload. These indexes were mainly clustered indexes created on primary key columns, and non clustered indexes created on foreign key columns for each of the tables. The T-SQL scripts of these indexes are provided in appendix B.

4.3.2 Experimentation with Query Parallel Processing

The second method of optimization of the performance of the system under test was the experimentation of the server settings involving the execution of queries in parallel. Comparing the response time of queries run with a serial execution plan against the response time of queries run with a parallel execution plan was among the objectives to be attained through this experiment.

Since Microsoft SQL Server 2008 is by default configured to use all the available processors of the multiprocessor architecture hardware platform on which it is installed, the results obtained with the server set to its default configurations were assumed to be the optimized ones. Experimenting with only one processor, two processors or three processors made available to SQL Server processes was then performed in order to establish a correlation between the number of processors used and the query response time observed. Each of the results set obtained were respectively compared with the results obtained with the database set to its default configurations. This was mainly done in order to experiment the effect of intra-query parallelism in SQL Server 2008. The same set of seven queries run with the experiment involving the indexes suggested by the tuning advisor tool were also used for this experiment but with only the database of scale factor =10 (10GB).

4.3.3 Parallel query processing – variable of interest

The main configurations allowing parallel query execution in Microsoft SQL Server 2008 are:

- The Max degree of parallelism
- The Cost threshold for parallelism
- The max worker threads

4.3.3.1. The Max degree of parallelism

This is a configuration variable that instructs to SQL Server the number of processor to be used during the execution of a particular query [26]. By defaults, SQL Server is configured to use all the available processors offered by the hardware platform on which is being run. Setting the Max degree of parallelism to 1, restricts SQL Server from generating any parallel execution plan since there will only be single processor made available for SQL Server processes. Different values of the Max degree of parallelism 1, 2, 3 and 0 were experimented since the hardware platform provides four processors.

4.3.3.2 The Cost Threshold for parallelism

The Cost threshold for parallelism specifies the elapsed time in seconds above which SQL Server stops generating a serial execution plan and starts generating a parallel execution plan. This happens when query optimizer estimates that the cost of a serial plan is higher than the one of a parallel plan for the same query. In this experiment The Cost Threshold for parallelism was left to its default configuration of five as recommended by Microsoft [26].

4.3.3.3 The Max worker threads

This configuration variable specifies the number of threads made available to the Microsoft SQL Server processes. It has the responsibility of creating a pool of worker threads meant to service a considerable number of query requests hence, improving the system's performance. The Max worker threads value was left to its default value of zero as recommended by Microsoft as being the best for most systems [28].

4.3 Chapter summary

This chapter aim was the description of the methodology adopted for the execution of every test performed during the experimental phase of this project. Specific variables of interest relevant to parallel query execution are also described. The next chapter focuses on the results obtained for each test performed.

Chapter 5-Results

This chapter is dedicated to the results gathered during the experimental phase of this project.

5.1 Performance measurements

The performance measurements were done using “SQL Server profiler” tool embedded in SQL Server which was used to trace all the SQL events that were taking place on the server. “Spotlight on SQL Server Enterprise” with its user interface provided an easy way of visualizing in real time the health of the system under test.

The performance measurements of interest were:

- The Average query response (in seconds) which is the amount of time it takes a single query to complete its execution run.
- The Average CPU time (in millisecond) which is the amount of the CPU spent on processing a given query
- The Average Disk Read which provides the average number of reads per second of data from the disk.
- The Average Disk write provides the average number of writes per second of data to the disk.

5.2 Experimental set-up

As already mentioned, the “Benchmark factory for databases version 6” software tool was used to generate three different databases of scale factor 1, 10, 30. The three databases as well as the operating system Microsoft Windows Server 2008 were left to their default configurations. Each of the twenty-two queries was then run against each of the three databases. The Microsoft SQL Server profiler tool was used to trace the time it took each query to execute, the average time it took the CPU to process each query, the average number of reads per second from the disk as well as the average number of writes per second to the disk that occurred during the execution of each query.

Though the benchmark factory for databases software provides a means of running queries against the database under test automatically after the loading of data into the database has completed, queries were run manually in order to compare the results produced from the execution of each query with the ones specified by the TPC-H specification manual provided in appendix since queries run automatically on the background using the benchmark factory for databases software do not produce any results. The Benchmark Factory for database software tool was just used to generate the workload against which the TPC-H queries were run manually.

5.3 Base line results – Databases set to their default configuration

The tables below provide the performance results for each of the twenty two queries run against the three databases: “TPCH_1GB” (scale factor=1), TPCH_30GB (scale factor=10), “TPCH_30GB” (scale factor=30GB)

5.3.1 Database of scale factor 1- “TPCH_1GB”

The table 6 below provides the performance results obtained after running a series of twenty two queries against the database “TPCH_1GB” generated with a scale factor of 1. Having approximately a size of 1GB, The largest table in this database which is the “LINEITEM” table has a cardinality of six million rows.

Transaction Results – Scale Factor 1 (1GB)				
Query Number	***Average Response Time (s)***	***Average CPU Time (ms)***	***Average Number of Reads (per ms)***	***Average Number of Writes (per ms)***
Q1	2.756	7816	119552	1
Q2	0.971	874	8357	4
Q3	0.781	2076	80134	0
Q4	0.873	2217	119530	0
Q5	0.999	2576	121061	0
Q6	0.481	702	22192	3
Q7	0.642	1559	60029	0
Q8	1.402	4285	62521	0
Q9	2.577	6505	137565	21
Q10	2.047	2682	123041	0
Q11	0.113	309	19380	0
Q12	0.385	1404	259539	0
Q13	1.425	3370	27831	20
Q14	0.132	404	5726	0
Q15	0.532	563	7746	0
Q16	0.333	983	6363	5
Q17	0.557	1264	9754	0
Q18	2.169	7285	116596	0
Q19	1.418	531	6360	0
Q20	1.156	1389	35726	2
Q21	2.622	6755	410683	12
Q22	0.511	732	19395	11

Table 6 - Baseline results with scale factor of 1

With a quick look at the table 6 above, it can be observed that the query response time differs for each and every query execution run. This can be explained by the complexity of each and every query as well as the number of disk reads or writes per second that need to be performed during the execution of that particular query in order to retrieve the expected results. A high number of disk reads per second can also be observed.

5.3.2 Database of scale factor 10 –"TPCH_10GB"

The table 7 below provides the performance results obtained after the execution of the twenty-two queries run against the database "TPCH_10GB" generated with a scale factor of 10. The largest table in this database is the "LINEITEM" table that has a cardinality of sixty million rows.

Transaction Results - Scale Factor 10 (10GB)				
Query Number	***Average Response Time (second)***	***CPU Average Time(ms)***	***Average Number Reads(per ms)***	***Average Number Write(per ms)***
Q1	932.122	93960	1160969	0
Q2	8.724	1950	46794	4
Q3	540.312	26740	778008	1
Q4	1007.875	29030	1185965	0
Q5	1049.707	71133	1212349	0
Q6	150.49	6397	185053	5
Q7	347.422	16349	598865	0
Q8	361.627	53132	912599	0
Q9	1039.455	80415	1357287	32
Q10	1064.473	25240	1211881	0
Q11	16.345	2480	177867	0
Q12	995.891	18721	3819258	0
Q13	43.266	37051	250550	22
Q14	14.103	3898	54315	0
Q15	75.977	5321	54496	2
Q16	15.476	9705	57599	18
Q17	0.985	1123	41951	0
Q18	1000.466	105891	1193145	2
Q19	42.17	5195	53287	9
Q20	99.573	3868	255463	7
Q21	3136.353	97235	5055285	16

Table 7-Baseline results with scale factor of 10

5.3.3 Database of scale factor 10 –"TPCH_10GB"

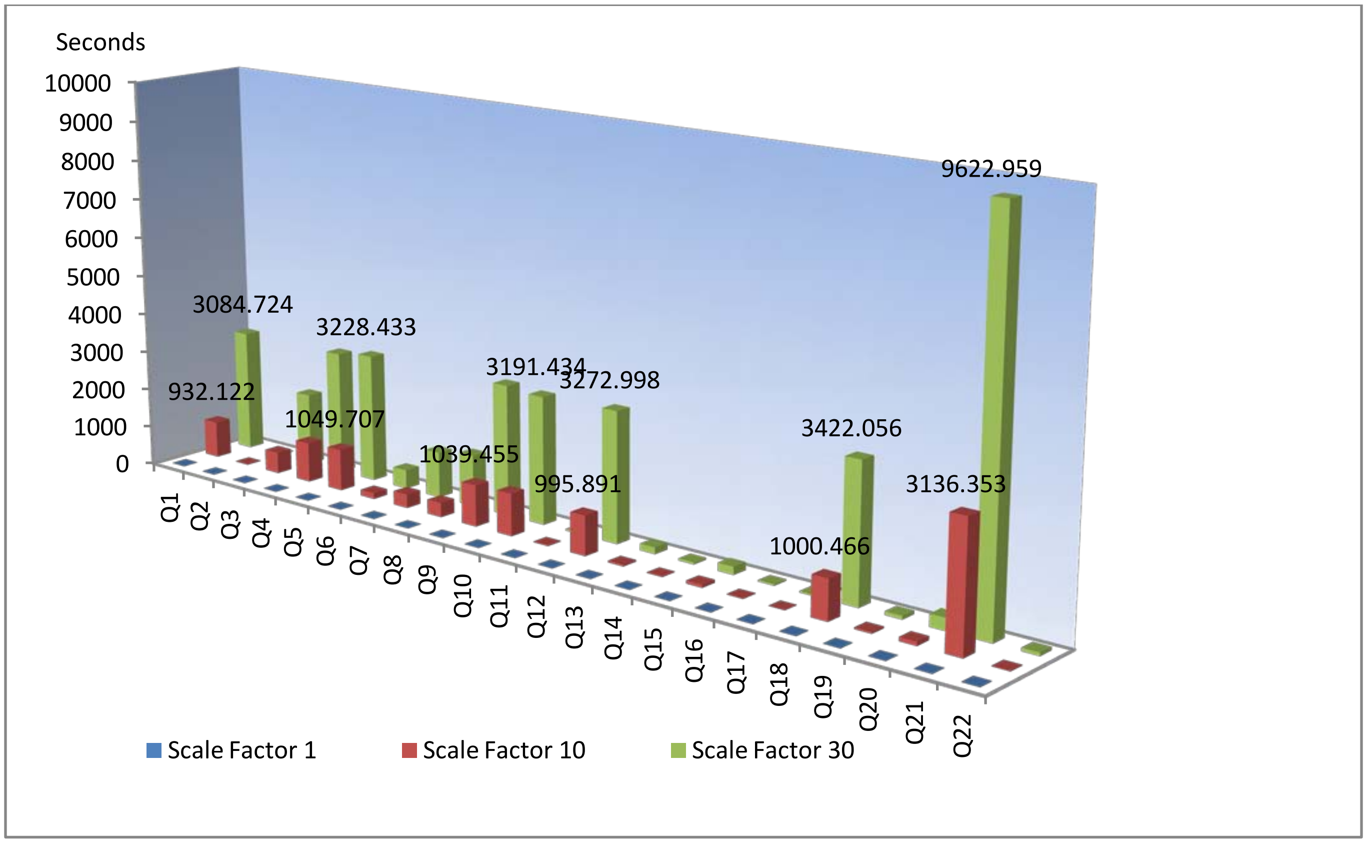
The table 8 below provides the performance results obtained after the execution of the twenty-two queries run against the database "TPCH_10GB" generated with a scale factor of 10. The largest table in this database is the "LINEITEM" table that has a cardinality of 180000000 rows.

Transaction Results – Scale Factor 30				
Query Number	***Average Response Time(s)***	***CPU Average time(ms)***	***Average of Read(per ms)***	***Average number of Writes(per ms)***
Q1	3084.724	293813	3466568	4
Q2	19.289	3102	129418	4
Q3	1814.337	89219	2330392	1
Q4	3106.35	86299	3549087	0
Q5	3228.433	249650	3682531	0
Q6	477.071	17440	545860	3
Q7	1158.625	52744	1797544	0
Q8	1302.811	165970	3040130	0
Q9	3270.1	256403	4124784	37
Q10	3191.434	79478	3649833	15
Q11	39.88	7565	533713	0
Q12	3272.998	55192	11326237	0
Q13	166.292	113881	745064	28
Q14	49.372	11761	157166	0
Q15	199.772	15959	156898	3
Q16	35.693	30138	169601	20
Q17	31.797	2559	119253	2
Q18	3422.056	357149	3736301	31
Q19	91.943	10422	139647	3
Q20	306.369	8192	1085089	15
Q21	9622.959	304884	11346020	26
Q22	82.483	13823	1345647	14

Table 8-Baseline results with scale factor of 30

5.3.4 Baseline results summary

The graph 1 below provides a summary of individual query response time obtained from the tables 6, 7 and 8 above.



Graph 1 - Query response time (summary)

As the scale factor increases, the response time of each individual query also increases depending on complexity of that particular query. We were particularly interested in queries that had the highest response time in the scale factor of 10 and 30 for the purpose of minimizing their execution time.

5.4 Selected queries

5.4.1 Long Running Queries

Q1, Q4, Q5, Q9, Q10, Q12, Q18 and Q21, were found to have long execution runs. Similar results were also obtained by Wasserman et al. who classified Q9, Q18, and Q21 as large and complex long running queries [24]. We chose to optimize queries Q4, Q5, Q9, Q10, Q12, Q18 and Q21 since they had the longest execution time in both scale factor 10 and 30.

The table 9 below provides response time of the selected queries in both scale factor 10 and 30.

	Scale Factor =10	*Scale Factor =30*
Query Number	***Average Response time(second)***	***Average Response Time(second)***
Q4	1007.875	3106.35
Q5	1049.707	3228.433
Q9	1039.455	3191.434
Q10	1064.473	3272.998
Q12	995.891	3272.998
Q18	1000.466	3422.056
Q21	3136.353	9622.959

Table 9-Long running queries

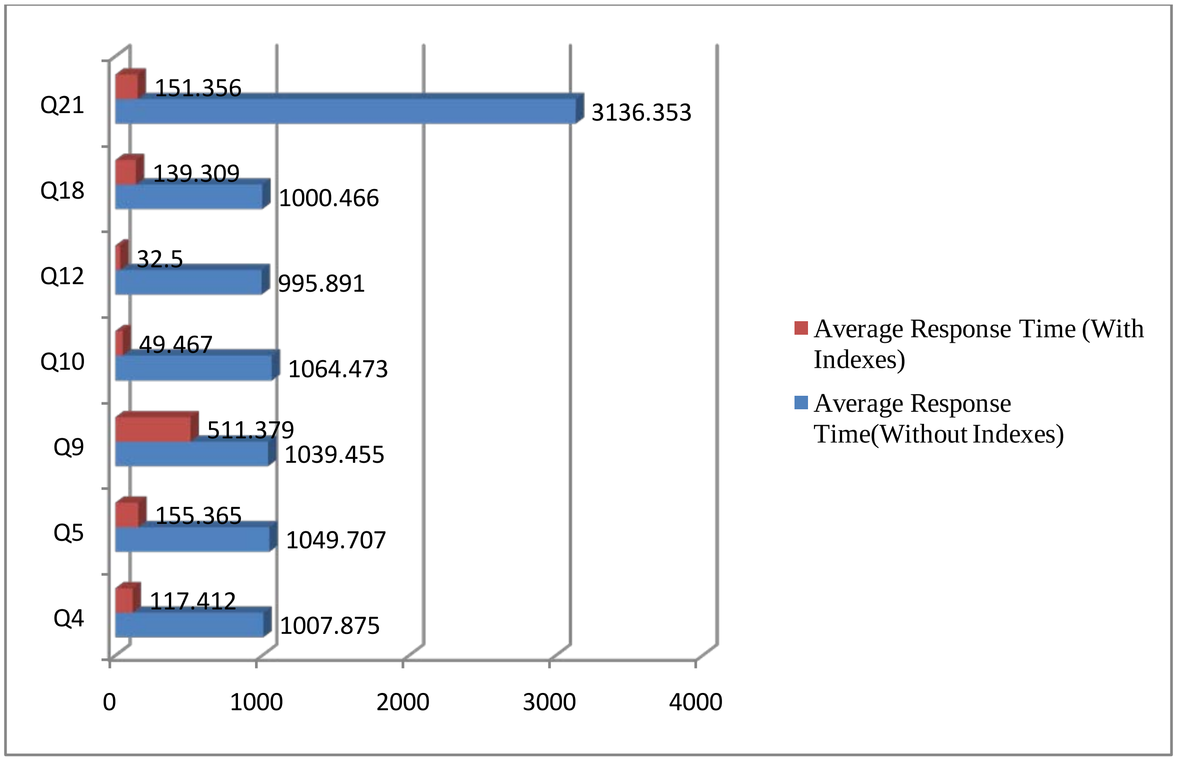
5.5 Query performance optimization – with indexes

For each of the two scale factors 10 and 30, each of the seven selected TPC-H query scripts (Q4, Q5, Q9, Q10, Q12, Q12, Q18, and Q21) was analyzed by the Tuning Advisor tool embedded in Microsoft SQL Server 2008 which then suggested indexes pertaining to query performance improvement.

This experiment aimed at the evaluation of the index suggestions provided by the Microsoft SQL Server Tuning advisor tool with respect to the decision support workload namely the TPC-H as the workload being used for this project.

5.5.1 Database with Scale Factor 10 – “TPCH_1GB”

The Graph 2 below provides a comparison of the response time obtained without indexes against ones obtained after creating the indexes generated by the Microsoft SQL Server Tuning Advisor tool.

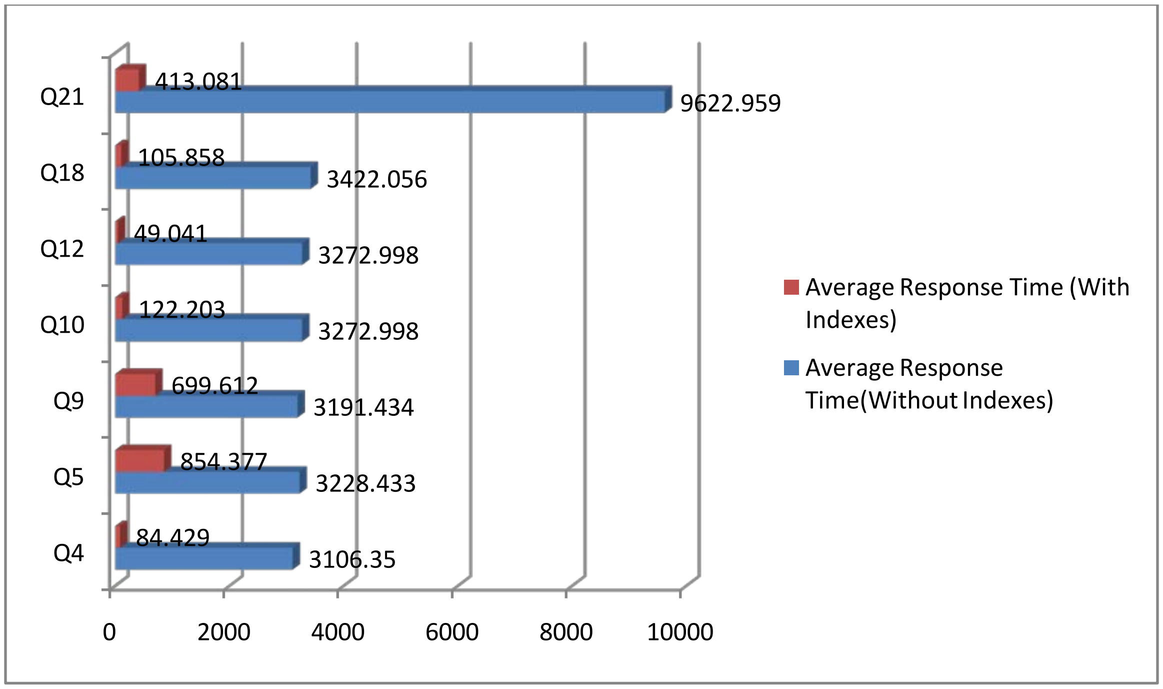


Graph 2 - Query response time with indexes vs query response without indexes(scale factor=10)

It can easily be observed from the Graph above that the execution time of queries run with indexes has significantly decreased as compared to the ones run without indexes (default settings).

5.5.2 Database with a Scale Factor of 30 – "TPCH_30GB"

The Graph 3 below provides a comparison of the response time obtained without indexes against ones obtained after creating the indexes generated by the Microsoft SQL Server Tuning Advisor tool.



Graph 3-Query response time with indexes vs query response time without indexes (scale factor=30)

It can easily be depicted from Graph above that the execution time of queries run with indexes has significantly decreased as compared to the one run without indexes (default settings).

5.5.3 Analysis of the suggested indexes

Since clustered indexes are created automatically when a primary key is created for a given table, the tuning advisor tool's suggestions are non clustered indexes. Knowing how indexes can improve the response time of a given query if they created on appropriate columns, an analysis of how the tuning advisor selects columns on which to create indexes under the TPC-H workload was performed. That is, how the Tuning advisor tool suggests indexes with respect to the TPC-H database schema, based on the complexity of the queries and the size of the database. The indexes scripts generated by the tuning advisor after processing queries: Q4, Q5, Q9, Q10, Q12, Q18, and Q21 served as the basis for the analysis. These scripts were the same for scale factor 10 and 30.

Each of the selected queries is designed to provide answers to specific business questions as defined by the TPC-H specification.

5.5.3.1 Query4 – Order Priority

- **Business question**

"The Order Priority Checking Query counts the number of orders ordered in a given quarter of a given year in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority sorted in ascending priority order."[30]

```
USE TPCH_10GB
SELECT o_orderpriority,
      COUNT (*) AS ORDER_COUNT
FROM H_Order

WHERE o_orderdate >= '1997-07-01'
     AND o_orderdate < DATEADD (mm, 3, cast ('1997-07-01' as SMALLDATETIME))

     AND EXISTS (SELECT *
                FROM H_Lineitem

                WHERE l_orderkey = o_orderkey
                     AND           l_commitdate < l_receiptdate
                )
GROUP BY o_orderpriority

ORDER BY o_orderpriority
```

Figure 4 - Query 4

Query 4 involves:

- A join operator between the H_Order and H_Lineitem tables on columns l_orderkey, o_orderkey, l_commidate and l_receiptdate.
- A Group by operator on column o_orderpriority and an order by operator on column o_orderpriority.

5.5.3.1.1 Query 4 Tuning Advisor suggested index

```
USE [TPCH_10GB]

GO
CREATE NONCLUSTERED INDEX [_dta_index_H_Lineitem_11_2137058649__K1_K12_K13]
ON [dbo].[H_Lineitem]
(
        [l_orderkey] ASC,
        [l_commitdate] ASC,
        [l_receiptdate] ASC
)
 WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Figure 5-Index suggested from Query 4

It can be observed from the figure 5 above that the tuning advisor suggested the creation of a non clustered index on columns "l_orderkey, l_commitedate and l_receiptdate involved in a join between tables H_Lineitem and H_Order. (Tables are defined in appendix A).

5.5.3.2 Query5 – Local Supplier Volume Query

- **Business question**

“The Local Supplier Volume Query lists for each nation in a region the revenue volume that resulted from lineitem transactions in which the customer ordering parts and the supplier filling them were both within that nation. The query is run in order to determine whether to institute local distribution centers in a given region. The query considers only parts ordered in a given year. The query displays the nations and revenue volume in descending order by revenue. Revenue volume for all qualifying lineitems in a particular nation is defined as
Sum (l_extendedprice * (1 -l_discount)).”[29]

```
USE TPCH_10GB
SELECT n_name,
      SUM (l_extendedprice * ( 1 - l_discount)) AS REVENUE
FROM    H_Customer, H_Order, H_Lineitem,   H_Supplier, H_Nation, H_Region

WHERE c_custkey = o_custkey
     AND l_orderkey = o_orderkey
     AND l_suppkey = s_suppkey
     AND c_nationkey = s_nationkey
     AND s_nationkey = n_nationkey
     AND n_regionkey = r_regionkey
     AND r_name = 'AFRICA'
     AND o_orderdate >= '1997-01-01'
     AND o_orderdate < DATEADD (YY, 1, cast ('1997-01-01' as SMALLDATETIME))

GROUP BY n_name
ORDER BY REVENUE DESC
```

Figure 6-Query 5

Query 5 involves:

- A join operator on column c_custkey and o_custkey between H_customer and H_orders tables, a join operator on column l_orderkey and o_orderkey between tables H_lineitem and H_Order, a join operator on columns l_suppkey and s_suppkey between H_Supplier and H_Lineitem, a join operator on columns c_nationkey and s_nationkey between tables

H_Customer and H_Supplier, a join operator on columns s_nationkey and n_nationkey between tables H_Supplier and H_Nation, and finally a join on columns n_regionkey and r_regionkey between tables H_Nation and H_Region.

- A “group by” operator on column n_name,
- An “order by” operator on the column revenue which is created by the aggregate operator “sum” involving columns: l_extendedprice and l_discount on H_lineitem table.

5.5.3.2.1 Query 5 Tuning Advisor suggested index

```
USE [TPCH_10GB]

GO

CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K3_K1_6_7] ON [dbo].[H_Lineitem]
(
	[l_suppkey] ASC,
	[l_orderkey] ASC
)
INCLUDE ( [l_extendedprice],
[l_discount])
WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
	SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING =
OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
GO
```

Figure 7-Index suggested from Query 5

The tuning advisor suggested a non clustered index on columns: l_orderkey and l_suppkey, that are involved in a join between H_Lineitem and H_Supplier tables as well as a join between H_Lineitem table and H_Order table It also contains included columns l_exentededpprice and l_discount.

5.5.3.3 Query9- Product Type Profit Measure Query

- **Business question**

"The Product Type Profit Measure Query finds, for each nation and each year, the profit for all parts ordered in that year that contain a specified substring in their names and that were filled by a supplier in that nation.
The profit is defined as the sum of [(l_extendedprice*(1-l_discount)) – (ps_supplycost * l_quantity)] for all lineitems describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first)."[30]

```
USE TPCH_10GB
GO
SELECT NATION,
       O_YEAR,
       SUM(AMOUNT)AS SUM_PROFIT
FROM (SELECT n_name NATION,
            DATEPART (YY, o_orderdate) AS O_YEAR ,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity AS AMOUNT
     FROM H_Part, H_Supplier, H_Lineitem, H_Partsupp, H_Order, H_Nation
     WHERE s_suppkey = l_suppkey AND ps_suppkey = l_suppkey
          AND ps_partkey = l_partkey AND p_partkey = l_partkey
          AND o_orderkey = l_orderkey AND s_nationkey = n_nationkey
          AND p_name LIKE '%almond%'
       ) AS PROFIT
GROUP BY NATION,
        O_YEAR
ORDER BY NATION,
        O_YEAR DESC
GO
```

Figure 8 - Query 9

Query 9 involves:

- A join operator on columns s_suppkey and l_suppkey between tables H_Supplier and H_Lineitem, a join operator on column ps_suppkey and l_suppkey between tables H_Partsupp and H_Lineitem, a join operator on columns ps_partkey and l_partkey between tables H_Partsupp and H_Part, a join operator on columns p_partkey and l_partkey between tables H_Part and H_lineitem, a join operator on columns o_orderkey and l_orderkey between tables H_Order and H_Lineitem, and finally a join operator on column s_nationkey and n_nationkey between tables H_Supplier and H_nation.
- A group by operator on n_name(nation) and o_year columns,
- An order by on n_name (nation) and o_year columns.

5.5.3.3.1 Query 9 Tuning Advisor suggested index

The tuning advisor suggested the creation of three non clustered indexes given as follows:

- A non clustered index on columns ps_partkey and ps_name columns based on H_part table, as given by the following script

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX [_dta_index_H_Part_11_5575058__K1_K2] ON
[dbo].[H_Part]
          (
          [p_partkey] ASC,
          [p_name] ASC
          )WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
          SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING =
OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  =
ON) ON [PRIMARY]
GO
```

Figure 9-Index suggested from Query 9

- A non clustered index on columns ps_partkey, s_suppkey both involved in a join operator and an included column ps_supplycost based on H_partsupp table as given by the following script.

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX
[_dta_index_H_Partsupp_11_2121058592__K1_K2_4] ON [dbo].[H_Partsupp]
          (
          [ps_partkey] ASC,
          [ps_suppkey] ASC
          )
INCLUDE ( [ps_supplycost]) WITH (PAD_INDEX  = OFF,
STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Figure 10 Index suggested from Query 9

- A non clustered index on columns: l_partkey, l_orderkey and l_suppkey that are involved in joins as well as an included columns l_quantity, l_extendedprice, l_discount as given by the following script.

```
USE [TPCH_10GB]
GO
CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K2_K1_K3_5_6_7] ON
[dbo].[H_Lineitem]
(
          [l_partkey] ASC,
          [l_orderkey] ASC,
          [l_suppkey] ASC
)
INCLUDE ( [l_quantity],[l_extendedprice],
[l_discount]) WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
GO
```

Figure 11-Index suggested from Query 9

5.5.3.4 Query10-Returned Item Reporting Query

- **Business question**

“The Returned Item Reporting Query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter. The query lists the customer's name, address, nation, phone number, account balance, comment information and revenue lost. The customers are listed in descending order of lost revenue.

Revenue lost is defined as sum(l_extendedprice*(1-l_discount)) for all qualifying lineitems”[29]

```
USE TPCH_10GB
GO
SELECT c_custkey,
       c_name, SUM (l_extendedprice * (1 - l_discount)) AS REVENUE,
       c_acctbal, n_name, c_address, c_phone, c_comment
FROM H_Customer, H_Order, H_Lineitem, H_Nation
WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey
      AND o_orderdate >= '1993-09-01'
      AND o_orderdate < DATEADD (MM, 3, cast ('1993-09-01' as SMALLDATETIME))

      AND l_returnflag = ‘R’ AND c_nationkey = n_nationkey
GROUP BY c_custkey,
        c_name, c_acctbal, c_phone, n_name,
        c_address, c_comment
ORDER BY REVENUE DESC
GO
```

Figure 12-Query 10

Query10 involves:

- A join operator on columns c_custkey, o_custkey between tables H_Customer and H_Order, a join operator on columns l_orderkey, o_orderkey between tables H_Lineitem and H_Order, as well as a join operator on columns c_nationkey and n_nationkey between tables H_Customer and H_nation.
- A group by operator on columns: c_custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment.
- An order by operator on column revenue which is created from the aggregate operator "sum" applied on column l_extendedprice and l_discount.

5.5.3.4.1 Query 10 Tuning Advisor suggested index

The tuning advisor suggested a non clustered index on l_returnflag and l_orderkey columns as well as an included columns l_extendedprice and l_discount as given by the following script.

```
USE [TPCH_10GB]

Go
CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K9_K1_6_7] ON [dbo].[H_Lineitem]
(
        [l_returnflag] ASC,
        [l_orderkey] ASC
)
INCLUDE ( [l_extendedprice],
[l_discount]) WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING =
OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
GO
```

Figure 13-Index suggested from Query 10

5.5.3.5 Query12 – Shipping Query Order Priority Query

- **Business question**

“This query determines whether selecting less expensive modes of shipping is negatively affecting the critical-priority
orders by causing more parts to be received by customers after the committed date.”[30]

```
USE TPCH_10GB
GO
SELECT l_shipmode,
       SUM (CASE  WHEN o_orderpriority = '1-URGENT'  OR o_orderpriority = '2-HIGH'
           THEN 1 ELSE 0
           END) AS HIGH_LINE_COUNT,
        SUM (CASE WHEN o_orderpriority <> '1-URGENT' AND o_orderpriority <> '2-HIGH'
            THEN 1 ELSE 0 END) AS LOW_LINE_COUNT
FROM H_Order, H_Lineitem
WHERE o_orderkey = l_orderkey AND l_shipmode IN ('REG AIR', 'FOB')
      AND l_commitdate < l_receiptdate AND l_shipdate < l_commitdate
      AND l_receiptdate >= '1997-01-01'
AND l_receiptdate < DATEADD (YY, 1, cast('1997-01-01' as SMALLDATETIME))
GROUP BY l_shipmode
ORDER BY l_shipmode
GO
```

Figure 14-Query 12

Query12 involves:

- A “Join” operator on columns o_orderkey and l_orderkey between tables H_order and H_lineitem,
- A “Group By” operator on column l_shipmode,
- An ‘order by” operator on column l_shipmode

5.5.3.5.1 Query 12 Tuning Advisor suggested index

The Tuning Advisor suggested indexes as follows:

- A non clustered index with columns o_orderkey and o_orderdate as well as an included column o_orderpriority based on table H_Order as given by the following script.

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX [_dta_index_H_Order_11_21575115__K1_K5_6] ON
[dbo].[H_Order]
(
	[o_orderkey] ASC,
	[o_orderdate] ASC
)
INCLUDE ( [o_orderpriority]) WITH (PAD_INDEX  = OFF,
STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY
= OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  = ON,
ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Figure 15-Indexes suggested from Query 12

- A non clustered index on columns l_shipmode, l_receiptdate, l_shipdate, l_orderkey based on table H_Lineitem as given by the following script

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K15_K13_K11_K1] ON [dbo].[H_Lineitem]
    (
        [l_shipmode] ASC,
        [l_receiptdate] ASC,
        [l_shipdate] ASC,
        [l_orderkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON
[PRIMARY]
GO
```

Figure 16-Index suggested from query 12

- A non clustered index on columns l_shipmode, l_receiptdate, l_shipdate, l_commitdate based on table H_Lineitem as given by the following script

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K15_K13_K11_K12] ON [dbo].[H_Lineitem]
    (
        [l_shipmode] ASC,
        [l_receiptdate] ASC,
        [l_shipdate] ASC,
        [l_commitdate] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
SORT_IN_TEMPDB      = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON
[PRIMARY]
GO
```

Figure 17-Index suggested by Query 12

5.5.3.6 Query18-Large Volume Customer

Business question

“The Large Volume Customer Query finds a list of the top 100 customers who have ever placed large quantity orders. The query lists the customer name, customer key, the order key, date and total price and the quantity for the order.” [30]

```
USE TPCH_10GB
GO
SELECT c_name,
      c_custkey, o_orderkey, o_orderdate, o_totalprice, sum (l_quantity)
FROM H_Customer, H_Order,   H_Lineitem
WHERE o_orderkey in (SELECT l_orderkey
                    FROM H_Lineitem
                    GROUP BY l_orderkey
                    HAVING sum (l_quantity) > 315
                    )
      AND c_custkey = o_custkey AND o_orderkey = l_orderkey
GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
ORDER BY o_totalprice DESC,
                     o_orderdate
 GO
```

Figure 18-Query 18

Query18 involves:

- A Join operator on columns c_custkey and o_custkey between tables H_Customer and H_Order and another join operator on columns o_orderkey and l_orderkey between tables H_Order table and H_Lineitem,
- An order by operator on columns o_totalprice and o_orderdate,
- A Group by operator on columns c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, l_orderkey,
- An aggregate operator on column l_quantity.

5.5.3.6.1 Query 18 Tuning Advisor suggested index

The tuning advisor suggested a non clustered index on column l_orderkey and contains included columns l_quantity and l_shipedate as given by the following figure:

```
USE [TPCH_10GB]

GO
CREATE NONCLUSTERED INDEX [_dta_index_H_Lineitem_11_2137058649__K1_5_11] ON
[dbo].[H_Lineitem]
          (
                    [l_orderkey] ASC
          )
INCLUDE ( [l_quantity],
[l_shipdate]) WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Figure 19 Index suggested from Query 18

5.5.3.7 Query21- Suppliers Who Kept Orders Waiting

- **Business question**

“The Suppliers Who Kept Orders Waiting query identifies suppliers, for a given nation, whose product was part of a multisupplier order (with current status of 'F') where they were the only supplier who failed to meet the committed delivery date.”[30]

```
USE TPCH_10GB
GO
SELECT s_name,
            Count (*) numwait
FROM H_Supplier, H_Lineitem l1, H_Order, H_Nation
WHERE s_suppkey = l1.l_suppkey  AND o_orderkey = l1.l_orderkey  AND o_orderstatus = 'F'
     AND l1.l_receiptdate > l1.l_commitdate
     AND EXISTS (SELECT *
              FROM H_Lineitem l2
WHERE l2.l_orderkey = l1.l_orderkey  AND l2.l_suppkey <> l1.l_suppkey
              )
     AND NOT EXISTS (SELECT *
                 FROM H_Lineitem l3
                 WHERE l3.l_orderkey = l1.l_orderkey AND l3.l_suppkey <> l1.l_suppkey
                 AND l3.l_receiptdate > l3.l_commitdate
                 )
      AND s_nationkey = n_nationkey AND n_name = 'ALGERIA'
GROUP BY s_name
ORDER BY numwait desc, s_name
GO
```

Figure 20-Query 21

Query 21 involves:

- A Join operator on columns s_suppkey and l_suppkey between tables H_Supplier and H_Lineitem, a second Join operator on columns o_orderkey and l_orderkey between tables H_Order and H_Lineitem, and third Join operator on columns s_nationkey and n_nationkey between H_Supplier and H_nation tables
- A "Group by" operator on column s_name,
- An Order by operator on columns s_name and numwait which is created from the aggregate operator "count",

5.5.3.7.1 Query21 Tuning Advisor suggested index

The tuning advisor suggested the following indexes:

- A Non Clustered index on columns: s_nationkey, s_suppkey, and s_name based on table H_Supplier, as provided by the index script below

```
USE [TPCH_10GB]
GO
CREATE NONCLUSTERED INDEX [_dta_index_H_Supplier_11_37575172__K4_K1_K2] ON
[dbo].[H_Supplier]
(
    [s_nationkey] ASC,
    [s_suppkey] ASC,
    [s_name] ASC
    )WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB
    = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
    ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Figure 21-Index suggested from Query 21

- A Non Clustered index on columns: o_orderstatus, o_orderkey and o_orderdate based on table H_Order,

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX [_dta_index_H_Order_11_21575115__K3_K1_K5] ON
[dbo].[H_Order]
            (
            [o_orderstatus] ASC,
            [o_orderkey] ASC,
            [o_orderdate] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
GO
```

Figure 22-Index suggested from Query21

- Non Clustered index on columns: l_orderkey, l_suppkey and l_receiptdate based on table H_Lineitem as provided by the index script below

```
USE [TPCH_10GB]
GO

CREATE NONCLUSTERED INDEX [_dta_index_H_Order_11_21575115__K3_K1_K5] ON
[dbo].[H_Order]
            (
            [o_orderstatus] ASC,
            [o_orderkey] ASC,
            [o_orderdate] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Figure 23-Index suggested from Query21

- A Non Clustered index on columns: l_suppkey, l_orderkey, l_receiptdate and l_commitdate based on table H_Lineitem as provided by the index script below

```
USE [TPCH_10GB]
 GO

CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K3_K1_K13_K12] ON [dbo].[H_Lineitem]
    (
    [l_suppkey] ASC,
    [l_orderkey] ASC,
    [l_receiptdate] ASC,
    [l_commitdate] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
 GO
```

Figure 24-Index suggested from Query21

5.5.4 Tuning Advisor suggested indexes evaluation

After the analysis of each individual index suggested by the tuning advisor, it can be observed that most of the columns involved in joins were chosen by the tuning advisor tool as being the columns on which to create Non Clustered indexes. Most of these columns are foreign keys on table on which the non clustered indexes are created. Some of the non clustered indexes contain included-columns which are non key columns of datatype that normaly are not supposed to be included in an index. Having more columns involved in joins in an index as well as included nonkey columns could justify how the query response time has greatly decreased. The reason being, the reading from the disk is reduced since the data required during the execution of a query can be found directly from the columns present in the index table instead of scanning the actual tables residing on the disk.

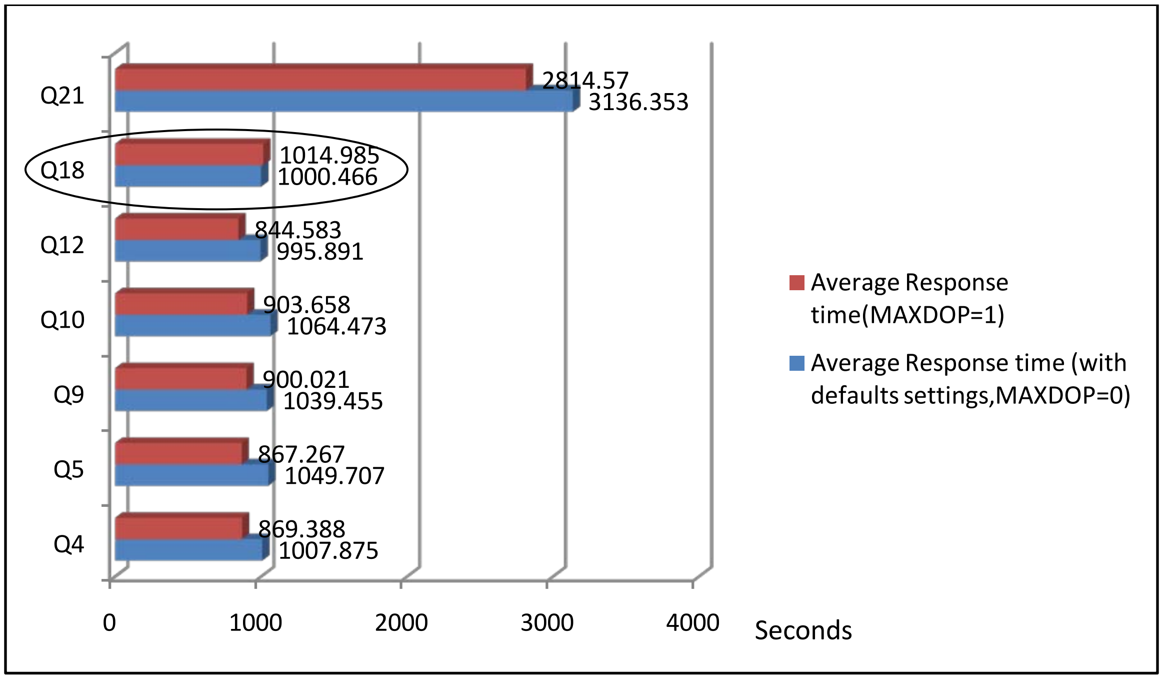
5.6 Query Performance optimization-with parallel query execution

The main objective to attain in executing queries in parallel was to take advantage of the multiprocessor architecture of the hardware platform consisting of four CPUs that were being used during experiment since SQL Server supports intra-operators parallel processing discussed in the Chapter 2 -Related work chapter of this thesis

The main idea is to experiment the variation in the query response time run with the default configurations (Max degree of parallelism set to 0, all the four CPUs are made available for the execution of the query). The database that was used for this experiment was the database of scale factor =10 (TPCH_10GB Database)

5.6.1 Max degree of parallelism set to 1

The graph 4 below compares query response time obtained with the Max degree of parallelism set to 1 with the query response time obtained with database set to its defaults configuration. By setting the Max degree of parallelism to 1, the query optimizer was forced to generate a serial query execution plan.

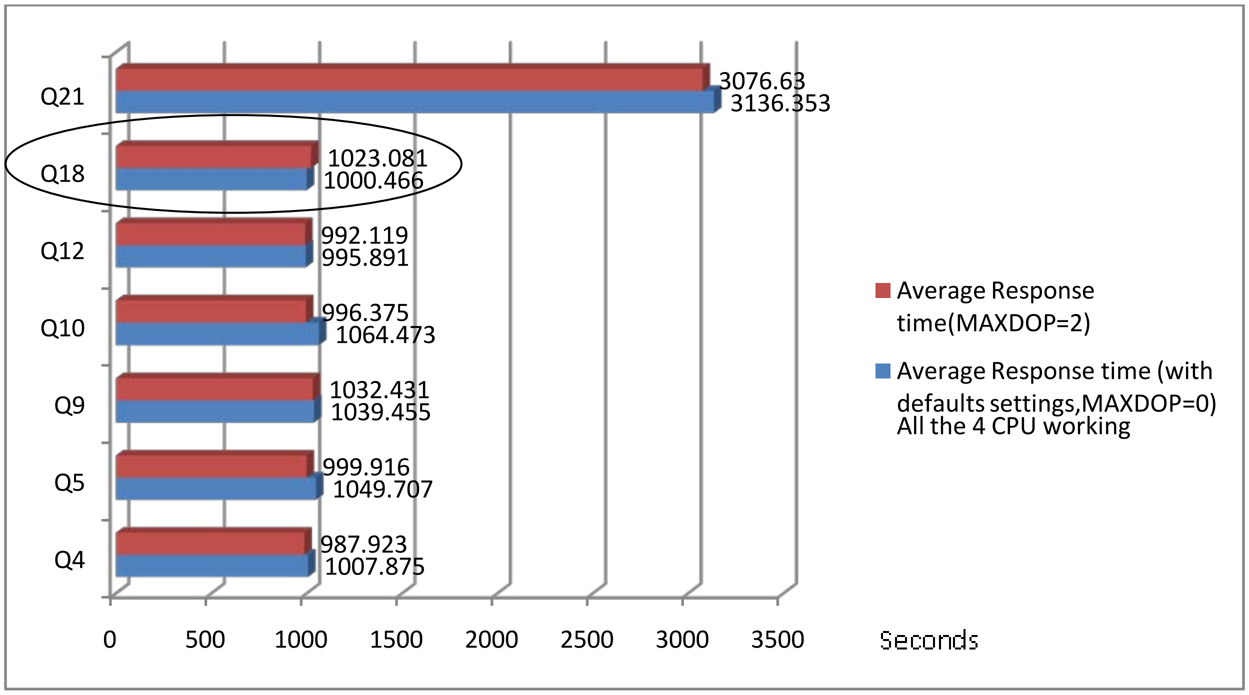


Graph 4- MAXDOP=1 vs MAXDOP=0 (Default configurations)

It can be observed that for all the query runs, the query execution time for the queries with the Max degree of parallelism set to 0 takes a bit longer to execute compared to the one with the Max degree of parallelism set 1 except for Query number 18. This goes against the performance improvement expected from executing complex and long running queries in parallel.

5.6.2 Max degree of parallelism set to 2

The graph 5 below compares the query response time obtained with the Max degree of parallelism set to 2 against the query response time obtained with the database set to its default configurations with the max degree of parallelism set 0 (All the four CPUs made available for SQL Server processes)

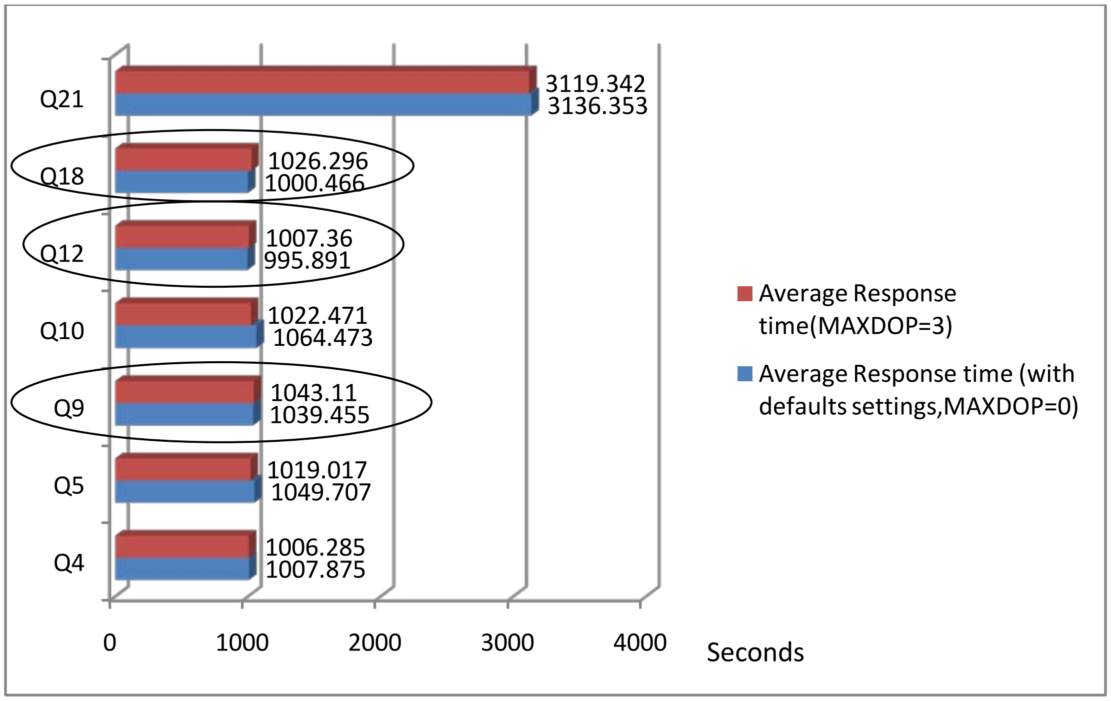


Graph 5-MAXDOP=2 vs MAXDOP=0 (Default configurations)

It can be observed from the Graph above that the execution run of each query with the database configured with the Max degree of parallelism set to 2 is slightly shorter than the one with the Max degree of parallelism set to 0 except for query Q18. From the results above it can be concluded that running the set of query above with two processors execute slightly faster that running them using four processors.

5.6.3 Max degree of parallelism set to 3

The graph 6 below compares query response time obtained with the Max degree of parallelism set to 3 with the query response time obtained with the database set to its default configurations with the Max degree of parallelism set to 0 (All the four CPUs are made available to SQL Server processes).



Graph 6-MAXDOP=3 vs MAXDOP=0 (Default Settings)

The graph above reveals slight differences between the execution run of queries with the Max degree of parallelism set to 3 and the response time obtained with the database set to its default configurations (MAXDOP=0). Q18, Q12 and Q9 executed slightly faster when run with four processors (MAXDOP=0) compared to their execution time when run with three processors but differences are not significant.

5.6.4 Results interpretation – Query parallel processing

The results obtained above appear to be counterintuitive to what was being expected. In order to understand the underlying problem causing such results, a close look at the SQL Server 2008 stored procedure "**sys.dm_os_wait_stats**" that provides information about all the waits encountered by the threads executed during the parallel execution of a given query needs to be done [20]. This stored procedure returns the:

- Wait_type : which is the name of the wait type,
- Waiting_tasks_count: which is the number of waits of a given type,
- Wait_time_ms: which is the total wait time for a particular wait type in milliseconds,
- Signal_wait_time_ms: which is the difference between the waiting time that the thread was signaled and when it started running.

The wait type that is of particular interest is the "CXPACKET" which occurs during the synchronization of the query processor exchange operators that are discussed in Chapter 2 – Related work (section 2.5.5.1) of this project.

The table 10 below provides the results that were observed after the execution of Query 4 against the database of size 10GB.

CXPACKET	Waiting_tasks_count	Wait_time_ms	Max_wait_time_ms	Signal_wait_time_ms
Q4(MAXDOP=1)	0	0	0	0
Q4(MAXDOP=2)	5199	3000981	998654	9309
Q4(MAXDOP=3)	6073	4153593	1036513	10096
Q4(MAXDOP=0) All the 4 CPUs running	5319	5006926	997765	70764

Table 10-Query4 run with different MAXDOP

The "sys.dm_os_wait_stats" stored procedure was cleared for each and every query execution run since the results returned accumulate for any SQL Server events running on the server. It can be observed that as the number of processors increases, the signal waiting time increases as well. A high number of waiting tasks and signal waiting time can be a good indicator of resource contention which can impact negatively on query response time as it can be observed with the results obtained above. There are indications that Microsoft may be addressing these problems. These indications are in the form of a recommendation that to overcome these problems the user should in the interim reduce the number of processors (i.e. set MAXDOP=1) [20]. There are also numerous blogs on the issue [1][14].

5.7 Chapter summary

The aim of this chapter was to provide the results obtained during the experimentation phase of project. The twenty-two TPC-H queries were run against three different databases of scale factor 1, 10, 30. A set of seven queries having the highest execution time in both the database of scale 10 and 30 were selected.

The first experiment consisted in submitting the selected queries to the tuning advisor tool embedded in SQL Server 2008 of analysis. The tuning advisor suggested indexes for each of the two databases. These indexes were then created on each of the two databases (scale factor of 10 and 30). Queries run after the creation of the tuning advisor suggested indexes executed faster than queries run without indexes. An in-depth analysis of the tuning advisor suggested queries revealed that the non clustered indexes suggested by the tuning advisor tool were created on appropriate columns such as foreign key columns and columns involved in joins.

The second experiment consisted in altering the database configuration settings involved in parallel query execution. It was observed that queries run sequentially executed faster than query in parallel. Queries with the database set to its default configuration (all the processors made available to SQL Server processes) took longer to execute compared to queries run with less than four processors.

Chapter 6 –Conclusion

As assigned at the beginning of this project, the objectives to be attained were an investigation of the TPC-H benchmark suite as the Transaction processing performance council for decision support systems, the explorations of different techniques used in performance optimization of the decision support systems as well as the application of some of these techniques in the performance optimization of Microsoft SQL Server 2008, 64 bit, Enterprise Edition run on Microsoft Windows Server 2008, 64 bit both installed on a 64 bit machine proline, with a core 2 quad CPUs at 2.66GHz each , 4GB of RAM and 500GB of hard disk.

The two techniques of optimizing the performance of decision support systems were experimented: The use of indexes as well as query parallel processing.

6.1 Findings

6.1.1 Experiment One: The use of indexes

The main objective to attain in using indexes was the analysis of indexes suggested by the tuning advisor tool embedded in Microsoft SQL Server 2008 on which the TPC-H workload was being run. A set of seven TPC-H queries (Q4, Q5, Q9, Q10, Q12, Q18, Q21) were run against two databases of size 10GB and 30GB left at their default configurations. Their response time was recorded. Furthermore, for each of the two databases, each of the selected queries were analyzed by the tuning advisor which then suggested Non clustered indexes scripts that were executed against both databases. A significant decrease in query response time was observed. The results of the analysis of indexes proved that the Tuning advisor created Non-clustered indexes on appropriate columns such as foreign key columns, columns involved in joins and where clauses, Group by and Order by operators are good candidates for Non clustered indexes (section 5.5). The suggested Non Clustered indexes also included non key columns allowing indexes to cover all the columns present in some of the queries hence, speeding up the retrieval of the data since it can be located directly from the index tables instead being fetch from the actual tables that might large to scan.

6.1.2 Experiment Two: Parallel query execution

Since SQL Server 2008 is configured to use by defaults all the available processors of the multiprocessor architecture hardware platform on which is installed, the main objective of altering the configuration settings pertaining to query parallelism was to compare the results obtained with all the four CPUs made available to SQL Server processes against the results obtained with only one CPU, two CPUs and three CPUs so as to establish a correlation between the query response time obtained with the number of CPUs made available to SQL Server processes. It was observed that queries run with a “max degree of parallelism” configuration variable set 1, only one CPU made available to SQL Server processes, executed faster than queries run with the server set to its default configurations (“max degree of parallelism” set to 0, i.e. 4 processors on a quad core computer). It was also observed that queries run with a “max degree of parallelism” variable set to 2, only two CPUs made available for SQL Server processes, executed faster than queries run with the server set to its default (max processors) configuration. The same facts were also observed with queries run with a he “max degree of parallelism set to 3”. However there were some queries that had a faster response time (than serial operation) with the server set to its default configurations but the differences were just slight.

6.2 Recommendations

Being characterized with a high number of disk reads activities, when used appropriately indexes appear to beneficial in the performance optimization of decision support workloads such as the TPC-H. The experiments conducted in project have proved that the tuning advisor tool embedded in SQL Server suggests indexes where appropriate. It is a tool to consider when planning the creation of indexes.

Query parallel processing on SQL Server 2008 needs to be examined carefully before being implemented on a production environment since queries run sequentially can execute faster than queries in parallel. The experiments conducted in this research again proved so.

6.3- Future Work

Some of the works that can be furthered with respect to this project are:

- The design of the transaction processing performance council benchmark software tool kits readily available for use on any operating systems platform that will easily allow database benchmark experimenters to compare their results with the published ones.

- The experimentation of performance optimization techniques such as the use of Redundant Array of Independent Disk and computer clusters.

- A comparison of parallel query processing between Microsoft SQL Server 2008 and other database products open source or proprietary.

References

[1]. Aaron Bertrand, Six reasons you should be nervous about parallelism http://sqlblog.com/blogs/aaron_bertrand/archive/2009/03/21/six-reasons-you-should-be-nervous-about-parallelism.aspx, 2009, [Accessed 01-10-2009]

[2]. Burgess G, What is the TPC Good For? Or, the Top Reasons in Favour of TPC Benchmarks, http://www.tpc.org/information/other/articles/TopTen.asp, 2009 [Access 17-06-2009]

[3]. Burleson, D., Database benchmarking, http://www.builderau.com.au/strategy/businessmanagement/soa/Database-benchmarking/0,339028271,320267276,00.htm, 2002, [Accessed 17-06-2009]

[4]. Coronel, C., *Database Systems: Design, Implementation & Management*, 5th Edition, Thomson Learning Inc, Massachusetts, USA, Inc., 2002

[5] Creating Indexes (Database Engine), http://msdn.microsoft.com/en-us/library/ms190197.aspx

[6]. Dam, S., *SQL Server Query Performance Tuning Distilled*, 2nd Edition, Appress, USA 2004

[7]. Darmont, J., Bentayeb, F., Boussaid, O., "Benchmarking data warehouses", *Inderscience Publishers*, Int. J. Bus. Intell. Data Min, 2007, Issue No 2, Vol 1 pages (79-104)

[8] Database Tuning Advisor Overview, http://msdn.microsoft.com/en-us/library/ms173494.aspx, [Accessed, 09-05-2009]

[9]. Dietrich, S., Brown, M., CORTES-RELLO, E., WUNDERLIN, S., "A Practitioner's Introduction to Database Performance Benchmarks and Measurements", *THE COMPUTER JOURNAL*, 1992, VOL.35, NO. 4

[10]. Floyd, C., Meikel, P., "New TPC benchmark for decision support and web commerce", *SIGMOD Record***,** Volume 29 Issue 4, ACM, December 2000

[11]. Graefe, G.,"Query Evaluation Techniques for Large Databases", *Computing Surveys (CSUR)*, ACM, 1993

[12] Huber, F. and Freytag J.C., "Query Processing on Multi-Core Architectures", http://rosdok.uni-rostock.de/file/rosdok_derivate_000000004041/gvd2009_2.A.04_Huber.pdf [Accessed 9-10-2009]

[13]. Hoste, K., Eeckhout, L., Blockeel H., "Analyzing commercial processor performance numbers for predicting performance applications of interest", *SIGMETRICS international conference on Measurement and modelling of computer systems*, San Diego, California, USA June 12–16, 2007

[14]. Journey to SQL Server Authority with Pinal Dave, http://blog.sqlauthority.com/2009/03/24/sql-server-2008-scope_identity-bug-with-multi-processor-parallel-plan-and-solution/, 2009, [Accessed, 03-10-2009]

[15] Max degree of parallelism option, http://msdn.microsoft.com/en-/library/aa196725(SQL.80).aspx [Accessed, 08-05-2009]

[16]. MicrosoftTechNet, Online Transaction Processing vs. Decision Support, http://technet.microsoft.com/en-us/library/ms187669.aspx, June 2009, [Accessed 10-07-2009]

[17] Microsoft SQL Server 2008, http://www.microsoft.com/sqlserver/2008/en/us/overview.aspx [Accessed, 31-05-2009]

[18] Microsoft Windows Server 2008, http://www.microsoft.com/windowsserver2008/en/us/default.aspx, [Accessed, 01-05-2009]

[19]. MSDN, SQL Server, Access Methods Object, http://msdn.microsoft.com/en-us/library/ms177426(SQL.90).aspx, November 2008, [Accessed 7-07-2009]

[20]MSDN, sys.dm_os_wait_stats (Transact-SQL), http://msdn.microsoft.com/en-us/library/ms179984.aspx, 2009, [Accessed, 03-07-2009]

[21]. Oracle, Database Benchmarking, http://wiki.oracle.com/page/Database+Benchmarking, 2009, [Accessed 16-06-2009]

[22]. Petkovic D., Microsoft SQL Server 2005, *A Beginner's Guide*, The McGraw-Hill Companies, California, USA, 2006

[23]. Scalzo, B., Benchmark Factory for Databases, http://www.quest.com/events/podcast/default.asp?path=/Quest_Site_Assets/podcasts/Quest_Software_-_Benchmark_Factory_-_Bert_Scalzo.mp3&title=Benefiting%20your%20IT%20Environment%20with%20Benchmark%20Factory, 2009, [Accessed 12-09-2009]

[24]. Scalzo B., Ault M., Burleson D., Fernandez C., Klein K., *Database Benchmarking, Practical Methods for Oracle & SQL Server*, Rampant TechPress, USA, April 2007

[25]. Shao, M., Ailamaki, A., Falsafi, B., "DBmbench: Fast and Accurate Database Workload Representation on Modern Microarchitecture", *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, ACM, 2005

[26]. SQL Server Book Online, Parallel Query Processing, http://msdn.microsoft.com/en-us/library/ms178065.aspx, October 2009, [Accessed, 27-06-2009]

[27]. SQL Server 2008 Books Online, Understanding Database Engine Tuning Advisor, http://technet.microsoft.com/en-us/library/ms188639.aspx, June 2009, [Accessed 11-07-09]

[28]. The Max worker threads option, http://msdn.microsoft.com/en-us/library/ms187024.aspx, [Accessed, 01-06-2009]

[29]. TPC, TPC Benchmarks, http://www.tpc.org/information/benchmarks.asp, 2009, [Accessed 15-06-2009]

[30]. TPC-H, http://www.tpc.org/tpch/default.asp, 2009, [Accessed 14-06-2009]

[31]. Using SQL Server Profiler, http://msdn.microsoft.com/en-us/library/ms187929.aspx, [Accessed, 10-05-2009]

[32]Webb Joe, Using covering indexes to improve query performance, http://www.simple-talk.com/sql/learn-sql-server/using-covering-indexes-to-improve-query-performance/, 28 September 2008, [Accessed 02-06-2009]

[33]. Wasserman, T.J., Martin, P., Rizvi, H., "Sizing DB2 UDB® Servers for Business Intelligence Workloads", *ACM*, 2004

[34]. Watts, D., Slavko, B., Watson, C., Understanding IBM eServer xSeries Benchmarks, http://www.redbooks.ibm.com/redpapers/pdfs/redp3957.pdf, 2005, [Accessed 3-06-09]

[35]. Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H., Urata, M., "Answering complex SQL queries using automatic summary tables", *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, ACM,

Appendix A: TPC-H Database- Table Layouts

Required Tables

The following list defines the required structure (list of columns) of each table. The annotations for primary keys and foreign references are for clarification only and do not specify any implementation requirement such as integrity constraints:

PART Table Layout

Primary Key: P_PARTKEY

Column Name	Data type Requirements	Comment
P_PARTKEY	identifier	SF*200,000 are populated
P_NAME	variable text, size 55	
P_MFGR	fixed text, size 25	
P_BRAND	fixed text, size 10	
P_TYPE	variable text, size 25	
P_SIZE	integer	
P_CONTAINER	fixed text, size 10	
P_RETAILPRICE	decimal	
P_COMMENT	variable text, size 23	

SUPPLIER Table Layout

Primary Key: S_SUPPKEY

Column Name	Data type Requirements	Comment
S_SUPPKEY	identifier	SF*10,000 are populated
S_NAME	text, size 25	
S_ADDRESS	variable text, size 40	
S_NATIONKEY	identifier	Foreign key reference to N_NATIONKEY
S_PHONE	fixed text, size 15	
S_ACCTBAL	decimal	
S_COMMENT	variable text, size 101	

PARTSUPP Table Layout

Compound Primary Key: PS_PARTKEY, PS_SUPPKEY

Column Name	Data type Requirements	Comment
PS_PARTKEY	identifier	key reference to P_PARTKEY
PS_SUPPKEY	identifier	Foreign key reference to S_SUPPKEY
PS_AVAILQTY	integer	
PS_SUPPLYCOST	decimal	
PS_COMMENT	variable text, size 199	

CUSTOMER Table Layout

Primary Key: C_CUSTKEY

Column Name	Data type Requirements	Comment
C_CUSTKEY	identifier	SF*150,000 are populated
C_NAME	variable text, size 25	
C_ADDRESS	variable text, size 40	
C_NATIONKEY	identifier	Foreign key reference to N_NATIONKEY
C_PHONE	fixed text, size 15	
C_ACCTBAL	decimal	
C_MKTSEGMENT	fixed text, size 10	
C_COMMENT	variable text, size 117	

ORDERS Table Layout

Primary Key: O_ORDERKEY

Column Name	Data type Requirements	Comment
O_ORDERKEY	identifier	SF*1,500,000 are sparsely populated
O_CUSTKEY	identifier	Foreign key reference to C_CUSTKEY
O_ORDERSTATUS	fixed text, size 1	
O_TOTALPRICE	decimal	
O_ORDERDATE	date	
O_ORDERPRIORITY	fixed text, size 15	
O_CLERK	fixed text, size 15	
O_SHIPPRIORITY	integer	
O_COMMENT	variable text, size 79	

LINEITEM Table Layout

Compound Primary Key: L_ORDERKEY, L_LINENUMBER

Column Name	Data type	Comment
L_ORDERKEY	identifier	Foreign key reference to O_ORDERKEY
L_PARTKEY	identifier	Compound Foreign Key Reference to (PS_PARTKEY, PS_SUPPKEY) with L_SUPPKEY
L_SUPPKEY	identifier	Foreign key reference to S_SUPPKEY, Compound Foreign key reference to (PS_PARTKEY, PS_SUPPKEY) with L_PARTKEY
L_LINENUMBER	integer	
L_QUANTITY	decimal	
L_EXTENDEDPRICE	decimal	
L_DISCOUNT	decimal	
L_TAX	decimal	
L_RETURNFLAG	fixed text, size 1	
L_LINESTATUS	fixed text, size 1	
L_SHIPDATE	date	
L_COMMITDATE	date	
L_RECEIPTDATE	date	
L_SHIPINSTRUCT	fixed text, size 25	
L_SHIPMODE	fixed text, size 10	
L_COMMENT	variable text size 44	

NATION Table Layout

Primary Key: N_NATIONKEY

Column Name	Data type Requirements	Comment
N_NATIONKEY	identifier	25 nations are populated
N_NAME	fixed text, size 25	Foreign key reference to R_REGIONKEY
N_REGIONKEY	identifier	
R_NAME	fixed text, size 25	
R_COMMENT	variable text, size 152	

REGION Table Layout

Primary Key: R_REGIONKEY

Column Name	Data type Requirements	Comment
R_REGIONKEY	identifier	5 regions are populated
R_NAME	fixed text, size 25	
R_COMMENT	variable text, size 152	

Appendix B: T-SQL Statements of indexes created before the Tuning Advisor indexes suggestions

```sql
CREATE CLUSTERED INDEX [H_lineitem_idx1] ON [dbo].[H_Lineitem]
(
	[l_shipdate] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO



CREATE NONCLUSTERED INDEX [H_lineitem_idx2] ON [dbo].[H_Lineitem]
(
	[l_partkey] ASC,
	[l_suppkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO


CREATE NONCLUSTERED INDEX [H_lineitem_idx3] ON [dbo].[H_Lineitem]
(
	[l_orderkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO



CREATE UNIQUE CLUSTERED INDEX [H_customer_idx1] ON [dbo].[H_Customer]
(
	[c_custkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO




CREATE NONCLUSTERED INDEX [H_customer_idx2] ON [dbo].[H_Customer]
(
	[c_nationkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

```
CREATE UNIQUE CLUSTERED INDEX [H_nation_idx1] ON [dbo].[H_Nation]
(
      [n_nationkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO



CREATE NONCLUSTERED INDEX [H_nation_idx2] ON [dbo].[H_Nation]
(
      [n_regionkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO


CREATE CLUSTERED INDEX [H_orders_idx1] ON [dbo].[H_Order]
(
      [o_orderdate] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO


CREATE NONCLUSTERED INDEX [H_orders_idx2] ON [dbo].[H_Order]
(
      [o_custkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO



CREATE UNIQUE NONCLUSTERED INDEX [H_orders_idx3] ON [dbo].[H_Order]
(
      [o_orderkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO


CREATE UNIQUE CLUSTERED INDEX [H_part_idx1] ON [dbo].[H_Part]
(
      [p_partkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

```
CREATE UNIQUE CLUSTERED INDEX [H_partsupp_idx1] ON [dbo].[H_Partsupp]
(
      [ps_partkey] ASC,
      [ps_suppkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO

CREATE NONCLUSTERED INDEX [H_partsupp_idx2] ON [dbo].[H_Partsupp]
(
      [ps_suppkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO


CREATE UNIQUE CLUSTERED INDEX [H_region_idx1] ON [dbo].[H_Region]
(
      [r_regionkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO




CREATE UNIQUE CLUSTERED INDEX [H_supplier_idx1] ON [dbo].[H_Supplier]
(
      [s_suppkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO



CREATE NONCLUSTERED INDEX [H_supplier_idx2] ON [dbo].[H_Supplier]
(
      [s_nationkey] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS  =
ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
GO
```

Appendix C: Server Configurations

Server Name: MYSERVER

Operating System: Microsoft Windows Server 2008

The Administrator password: Samy_2009_01

Database Management system product: Microsoft SQL Server 2008

The three databases tested can be accessed directly using the Microsoft SQL Server Management Studio graphical interface with the following details:

- Server type: Database Engine
- Server name: MYSERVER
- Authentication: Windows Authentication

The three databases tested are named:

- TPCH_1GB
- TPCH_10GB
- TPCH_30GB