ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 8(27), DOI: 10.17485/ijst/2015/v8i27/70620, October 2015* ISSN (Online) : 0974-5645

Novel Investigation Methodologies to Identify the SQL Server Query Performance

Muthukumar Murugesan[1*] , K. Karthikeyan[2] and K. Sivakumar[3]

[1]Department of Application Delivery, Mphasis Limited, Bangalore - 560048, Karnataka, India; amgmuthu@yahoo.com
[2]Department of Computer Application, Anna University, Madurai - 625007, Tamil Nadu, India; adithyakarthi@gmail.com
[3]Department of Computer Science and Electronics, Hindusthan Institute of Technology, Coimbatore - 641032, Tamil Nadu, India; rksivakumar@gmail.com

Abstract

Objectives: Performance optimization is an ever end process. It requires continuous monitoring and special attention. Widely known fact that 60% of the performance problems are direct result of the inefficient queries. **Method of Analysis:** SQL Query is one of the most essential parts of application performance. Over the period, inefficient queries pull down the performance in live applications gradually. To achieve better application performance, proper database design and efficient query are needed. Most of the database systems provide expected performance in early stage and drastically will come down, once the data volume is increased and database becomes large size. **Findings:** Real performances of database systems can be assessed only during the applications are in live Database administrator and query developers cannot make sure 100% performance issues during the database design and query writing. Lot of issues will come mostly in post deployment in particular, when database has heavy transactions per day. There are multiple approaches available in the market to provide the solution for database performance. But, none of the mechanism is available to find out the performance bottleneck proactively. Also, solving performance issues is very painful and time-consuming. **Improvement:** This paper proposes various novel approaches to identify the performance bottleneck upfront and demonstrates the solution for the same.

Keywords: Database Performance, Performance Bottleneck, Query Performance, Query Tuning, SQL Server Database

1. Introduction

Global providers like insurance, banking, finance, railway system and others have massive number of transactions on daily basis. All these transactions should be captured in the database for further reference. Over the period, size of these databases will get increased to MBs and GBs of data. Most of the complex queries will give performance issues, when they process enormous number of records.

SQL developers and database administrators should possess special skills and techniques to improve the query performance. Even for a powerful infrastructure, the performance can be significantly degraded by inefficient queries[3]. The ideas and processes of this paper will help to avoid these situations in future. Fortunately, it is not needed to have a lot of SQL experience. Identifying these bottle necks is not that difficult, if it is handled with a plan and few secrets. This paper is focuses on a plan and few secrets, and they are

- Describing protocols that can be used to find and solve SQL performance issues.
- Details of specific techniques that can be used to discover the causes of SQL performance issues and solve them.

Along the way, it can be understood that SQL server commands can help to become self-sufficient[5]. The steps

* *Author for correspondence*

and approaches available in this paper are very similar, and can be applied to other environment without any major challenges.

2. Significance of Identifying Performance Bottleneck

End users will get frustration, when the application has performance issue[3,4]. During this time, there won't be any idea on where to begin. Identifying the performance bottleneck proactively will increase the application performance and save lot time and cost. The novel approaches available in this paper will provide the solutions for:

- The people who are facing the performance issue on existing applications.
- When SQL statements take forever to run. Or worse, they run for so long (sometimes hours).
- The people who are planning to increase their business transaction massively.

Before planning to increase the business transaction[6,8], it should be made sure on performance and scalability of the existing application. Based on the volume of database, present database transactions will forecast the application performance and scalability for the future enhancement. Proactively doing this way, the performance bottleneck can be identified and existing SQL queries can be tuned. As a result, lot of future issues can be avoided[2] and also end users may be made happy. This proposed unique approach will help the administrator to diagnose problems, tune the server for optimal performance and troubleshoot the SQL queries.

3. Experimental Configuration

To enable better performance, a good system configuration and correct database software are needed. System configuration depends on the system throughput, system stability and user maintenance ability.

3.1 Software Configuration

For the experimental analysis, SQL server 2012 database is used. It is widely used in all large scale industries and data warehousing. In order to measure the performance bottleneck, sample real time database, which has massive number of transaction per day is used. The information analyzing is available as long as SQL Server is running. Once SQL Server is stopped, all the information collected will get lost. Statistics collected in this paper is based on query plan cached data which are available in SQL Server.

3.2 System Configuration

All the tests are executed on a physical machine Intel(R) Core™ i5 CPU @ 2.5 GHz Processer with 64 bit operating system, hosted on a computer with Windows 7 Enterprise with Service Pack 1 with a total of 4GB RAM. It is important to notice that Microsoft SQL Server 2008 R2 with 64 bit is used for the database.

4. Experimental Evaluation

The following sub section demonstrates how to start and where to begin to identify the performance bottleneck. The SQL server is tested using a mix of real time examples to better understand the various approaches and how the performance is affected internally by each complex and inefficient queries. The different methodologies and the benchmark of the query execution are measured in terms of

- Most Expensive Queries.
- Most Frequently Executed Queries.
- Most I/Os per Executions.
- CPU and Memory Utilization and Availability.

By default, SQL Server will store all the information related to query execution and database transactions in internal system tables. SQL Server dynamic management views[14–16] will give almost all the details related to each and every query execution. Dynamic Management Views (DMVs) gives internal information about SQL Server instance and database state and also they are useful for performance monitoring and troubleshooting. SQL Server system database (Master), holds all these schema related information.

4.1 Most Expensive Queries

This section provides detailed information about what are all the queries take more time to consume the CPUs and what are all the most expensive queries available in the database. CPU utilization has been calculated based on the query completion in seconds.

The SQL statement provides all information needed to find most expensive queries currently running, or

recently executed where their execution plan is still in the cache[14,15]. They can only show queries that are in the cache. Hence, if there is an expensive query, that's replaced in the cache by a new one. The result set will not show it. For experimental purpose, here only top 10 most expensive queries are considered. This number can be changed to any number based on the business requirement.

```
SELECT top 10
[Avg_CPU_Utilized_in_Seconds] =(total_worker_time * 1.0 / qs.execution_count * 1.0)/1000000,
[CPU_Utilized_in_Seconds] = total_worker_time * 1.0/1000000,
[Execution count] = qs.execution_count,
[Individual Query] =SUBSTRING(qt.text,qs.statement_start_offset/2,
(CASEWHEN qs.statement_end_offset =-1
THENLEN(CONVERT(NVARCHAR(MAX), qt.text))* 2
ELSE qs.statement_end_offset END-
qs.statement_start_offset)/2)
,[Parent Query] = qt.text
,[Last Execution Time] = qs.last_execution_time
FROMsys.dm_exec_query_stats qs
CROSSAPPLYsys.dm_exec_sql_text(qs.sql_handle) as qt
WhereDB_NAME(qt.dbid)= 'Employee'
ORDERBY [Avg_CPU_Utilized_in_Seconds] DESC;
```

Execution results of the above query have been captured in Figure 1. From the below figure, the average CPU utilized time, CPU utilized time, execution count, query which involved, parent query or stored procedure involved and query execution time are determined. Here, average processor time has been calculated based on total time the processor used to execute a query and is divided by the number of query executions, i.e. total worker time/ execution count.

From the above figure:

- First record gives details of top most expensive query.
- This UPDATE query is executed three times and consumed CPU 21.303 seconds as an average.
- Column Individual Query gives the actual query text involved in the execution.
- Column Parent Query gives the name of the database objects (stored procedure, views and triggers) where the actual SQL query exists.
- Last column gives the information of the date and time of the actual query executed last.

4.2 Most Frequently Executed Queries

Identifying most frequently executed queries is one of the most important factors while finding out the performance bottleneck. This section provides detailed information about most frequently executed queries. This doesn't indicate a slow query and will give the information of the statement executed many times. Also, this includes query execution plan as well. Query execution plan[10,11] is very useful for understanding the performance characteristics of a query. It calculates in most efficient way to implement the request represented by the T-SQL query that has been submitted. Execution plan tell show a query is executed and identifies the exact piece of SQL code that causes the problem. When a SQL query takes a long time to complete, execution plan will help to determine where the tuning is required.

The below SQL statement provides the details about most frequently executed and recent queries. The

	Execution count	Individual Query	Parent Query	query_plan	Last Execution Time
1	23825	SELECT @ApplicationId = ApplicationId FROM aspnet...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_GetRol...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-05-01 00:22:31.977
2	23825	SELECT @UserId = UserId FROM dbo.aspnet_Us...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_GetRol...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-05-01 00:22:31.977
3	13876	SELECT r.RoleName FROM dbo.aspnet_Roles r, d...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_GetRol...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 16:51:30.677
4	2342	SELECT @slice=@slice+' => ' + LevelName FROM LE...	CREATE FUNCTION ScalarSplitString (@Input N...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 08:34:04.030
5	1504	SELECT @ApplicationId = ApplicationId FROM aspnet...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_IsUserI...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 12:21:17.447
6	1504	SELECT @UserId = UserId FROM dbo.aspnet_Us...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_IsUserI...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 12:21:17.447
7	516	IF (EXISTS(SELECT * FROM dbo.aspnet_...	CREATE PROCEDURE [dbo].aspnet_CheckSchemaVersio...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-05-01 00:12:07.290
8	135	SELECT @RoleId = RoleId FROM dbo.aspnet_Ro...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_IsUserI...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 06:26:29.020
9	135	IF (EXISTS(SELECT * FROM dbo.aspnet_UsersInRole...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_IsUserI...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 06:26:29.020
10	130	SELECT @ApplicationId = ApplicationId FROM aspnet...	CREATE PROCEDURE dbo.aspnet_UsersInRoles_GetRol...	<ShowPlanXML xmlns="http://schemas.microsoft.com...	2015-04-30 15:30:57.643

Figure 1. Most expensive queries.

returned result set contains a row for every statement in the cached plan. For experimental purpose, here only top 10 most expensive queries are considered. This number can be changed to any number based on the business requirement.

```
SELECT TOP 10
 [Execution count] = qs.execution_count,
    [Individual Query] =SUBSTRING(qt.text,qs.
statement_start_offset/2,
 (CASEWHEN qs.statement_end_offset =-1
 THENLEN(CONVERT(NVARCHAR(MAX),
qt.text))* 2
 ELSE qs.statement_end_offset END-
 qs.statement_start_offset)/2)
 ,[Parent Query] = qt.text
 ,deqp.query_plan
 ,[Last Execution Time] = qs.last_execution_time
 FROMsys.dm_exec_query_stats qs
 CROSSAPPLYsys.dm_exec_sql_text(qs.sql_handle)
as qt
 CROSSAPPLY(SELECT*
 FROMsys.dm_exec_query_plan(qs.plan_handle))
deqp
 WhereDB_NAME(qt.dbid)='Employee'
 ORDERBY [Execution count] DESC;
```

- Fourth column gives the detailed information about query execution plan.
- Last column gives the information of the date and time of the actual query executed last.

4.3 Most Input/Output (I/O) per Executions

Identifying most I/O queries is also one of the most important factors finding out the performance bottleneck. The performance of applications is inherently limited by disk input/output (I/O). Often, CPU activity must be suspended, while I/O activity completes. SQL query has to be designed so that the performance is not limited by I/O. By looking at instance and identifying the databases[6,7] that are using the bulk of the I/Os will help to identify where to focus the tuning effort. By drilling into the high I/O databases and finding the specific T-SQL statements that consume large amounts of I/Os, those problem queries can be quickly identified. By reducing I/O on big I/O consuming queries and databases, it can be helped to tune-up the proposed server from an I/O perspective.

There is a single application using a single database, or multiple applications running against multiple databases on the same instance. However, for multiple databases running on an instance, it must be determined that how much I/O each database is generating. Based on this, count will decide which database and queries are using most of the I/Os. The SQL statements below will help to return the I/O statistics associated with the DATA and LOG files for all databases on an instance.

	Average IOs	avg_logical_reads	avg_logical_writes	avg_phys_reads	execution_count	query_text
1	3000173	2993690	6483	0	3	UPDATE [Emp_Master] SET Name=UT.Name,Email=UT.Em...
2	1364389	1348263	15951	174	3	insert into Emp_Master_Main(empid,name,email,nt_id,status,...
3	712580	707508	5072	0	1	Delete from Emp_Master_Main where NT_ID = ''
4	379484	372899	6585	0	3	UPDATE #UserTable SET IsEmpIdExist = 1 FROM [Emp_M...
5	237793	237315	70	408	14	INSERT INTO @tblProductSeriesGlb SELECT PS.Product...
6	164947	164495	452	0	1	SELECT 1 AS [C1], [Extent1].[EMPID] AS [EMPID], ...
7	164447	163900	547	0	1750	SELECT 1 AS [C1], [Extent1].[EMPID] AS [EMPID], ...
8	163581	163036	545	0	1	SELECT 1 AS [C1], [Extent1].[EMPID] AS [EMPID], ...
9	84589	84589	0	0	1	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp....
10	84457	84457	0	0	2	SELECT SCHEMA_NAME(sp.schema_id) AS [Schema], sp....

Figure 2. Most frequently executed queries.

Results of the above query execution have been captured in the Figure 2. From the below figure, execution count, actual query which involved on maximum times, parent query or stored procedure involved and query execution time will be made sure.

From the above figure:

- First column gives the count about number of times a query has executed.
- Columns second and third give the actual query text involved and parent database object (stored procedure, views and triggers) where the actual query exists respectively.

Table 1. Most I/O usages database

Database Name	Total I/Os
Employee	187000
Temp db	114843
Payroll	9726
Enterprise Directory	2307
Ms db	1846

```
SELECTTOP 5 DB_NAME(database_id)AS [Database Name]
,SUM(num_of_reads + num_of_writes)AS [Total I/Os]
FROMsys.dm_io_virtual_file_stats(NULL,NULL)
GROUPBY database_id
ORDERBYSUM(num_of_reads + num_of_writes) DESC
```

For experimental purpose, here only top 5 most I/O usages databases are considered. This number can be changed to any number based on the business requirement. The Table 1 depicts the results of the above query. The Table 1 also gives the high level idea of from where further investigation can be started. It means which database has highly used I/O. Further, analysis can be started based on the particular database.

Now, it is known that which database is issuing the bulk of the I/O operations. Then, the next step is to identify which T-SQL statements within that database are issuing majority of I/Os. Eventually, this will help to fine-tune a few queries to reduce the database I/Os considerably. The T-SQL statement that identifies the 10 most expensive T-SQL statements from an I/O perspective that are executed from the "master" database are given below:

```
SELECTTOP 10
(total_logical_reads + total_logical_writes +total_physical_reads )/
 execution_count [Average IOs],
(total_logical_reads/execution_count)AS avg_logical_reads,
(total_logical_writes/execution_count)AS avg_logical_writes,
(total_physical_reads/execution_count)AS avg_phys_reads,
(SELECTSUBSTRING(text, statement_start_offset/2 + 1,
(CASEWHEN statement_end_offset =-1
THENLEN(CONVERT(nvarchar(MAX),text))* 2
ELSE statement_end_offset
END- statement_start_offset)/2)
FROMsys.dm_exec_sql_text(sql_handle))AS query_text
FROMsys.dm_exec_query_stats
ORDERBY [Average IOs] DESC;
```

The above SQL statement displays a number of different statistics associated with the master database. The results of the above statement show statistics order by the statement that performs the most I/Os on average. From the above Figure 3:

- First column "Average IOs" has been calculated based on dividing the value of execution count with sum of "total logical reads, logical writes and physical reads".
- Next, three columns give the details about total logical reads, total logical writes and total physical reads.
- Fifth column gives the count about number of times a query has been executed.
- Last column gives the actual query text involved in I/O operation.

4.4 CPU and Memory Utilization and Availability

CPU and memory utilization are the most important factors in performance bottleneck[1,2]. Each system has some maximum capacity to process the SQL server request simultaneously. When SQL server sends the request constantly beyond the server maximum capacity, server may use 100% CPU or memory and automatically server virtual memory will get filled. Then, the server will have to restart its SQL server instance periodically. During this critical situation, the CPU and memory utilization have to be monitored periodically to identify which query or store procedure causes the issue. For example, assume current SQL server can process 100-

	ID	EventTime	Total_Memory_GB	Usage_Memory_GB	Available_Memory_GB	Memory_Usage	CPU_OtherProcess_Utilization	CPU_SQLProcess_Utilization	CPU_Idle	CreatedDate
9	74	2015-05-02 17:32:41.447	3.95	3.57	0.38	90.38	26	0	74	2015-05-02 17:32:46.513
10	81	2015-05-02 17:32:41.447	3.95	3.59	0.36	90.79	26	0	74	2015-05-02 17:32:54.013
11	73	2015-05-02 17:30:40.483	3.95	3.73	0.22	94.47	23	0	77	2015-05-02 17:31:10.840
12	72	2015-04-30 17:58:41.437	3.95	2.79	1.16	70.54	9	0	91	2015-04-30 17:58:47.167
13	69	2015-03-17 15:48:00.970	3.95	3.05	0.90	77.21	8	0	92	2015-03-17 15:48:48.283
14	68	2015-03-15 18:38:29.817	3.95	2.46	1.49	62.29	11	1	88	2015-03-15 18:38:40.707
15	67	2015-03-15 18:38:29.813	3.95	2.46	1.49	62.28	11	1	88	2015-03-15 18:38:30.573
16	65	2015-03-15 18:37:29.700	3.95	2.47	1.48	62.5	11	0	89	2015-03-15 18:38:10.307
17	66	2015-03-15 18:37:29.700	3.95	2.46	1.49	62.33	11	0	89	2015-03-15 18:38:20.880
18	61	2015-03-15 18:37:29.697	3.95	2.46	1.49	62.35	11	0	89	2015-03-15 18:37:30.403
19	62	2015-03-15 18:37:29.697	3.95	2.46	1.49	62.36	11	0	89	2015-03-15 18:37:40.523
20	63	2015-03-15 18:37:29.697	3.95	2.46	1.49	62.38	11	0	89	2015-03-15 18:37:50.627
21	64	2015-03-15 18:37:29.697	3.95	2.47	1.48	62.63	11	0	89	2015-03-15 18:38:00.243
22	56	2015-03-15 18:36:29.607	3.95	2.44	1.51	61.91	11	0	89	2015-03-15 18:36:40.800

Figure 3. Most I/O usages queries.

150 queries per second, if the amount even 1 higher than the expected is set, it may cause the issue either in CPU or memory. Though, the increasing capacity of CPU or memory will not help much.

High CPU utilization can be masking a number of other application or hardware bottlenecks too. Once it is identified with high CPU utilization, despite other counters looking healthy, it can be started looking for the cause within the system. In[2,3] high level, there are two paths to identify CPU performance problems. The first is reviewing the performance of system hardware and second one is reviewing the server's query efficiency. Second path is usually more effective in identifying SQL Server performance issues. Unless it is known exactly where the query performance issues lie, however, it should be always started with a system performance evaluation.

Knowing where to look for trouble is important, but more crucial is identifying why the system reacts the way that it does to a particular request. A number of factors can affect CPU utilization on a database server: compilation and recompilation of SQL statements, missing indexes, multithreaded operations, disk bottlenecks, memory bottlenecks, routine maintenance, extract, transform, and load (ETL) activity, among others. The key to healthy CPU utilization is making sure that the CPU is spending its time processing what it is wanted to process and not wasting cycles on poorly optimized code or sluggish hardware.

The below SQL statement will help to determine where to look when the first path is headed down.

```
DECLARE
@memory_usage FLOAT= 0.0
, @tot_memory DECIMAL(19,2)= 0.0
, @available_memory DECIMAL(19,2)= 0.0
, @Usage_memory DECIMAL(19,2)= 0.0
, @cpu_usage FLOAT= 0.0
, @CPU_Utilization VARCHAR(100)= 0.0
, @COUNT INT=1

SET @tot_memory =(SELECT (CAST(total_physical_memory_kb ASDECIMAL(22,3))/1024)/1024 FROMsys.dm_os_sys_memory)

SET@available_memory=(SELECT (CAST(available_physical_memory_kb ASDECIMAL(22,3))/1024)/1024 FROMsys.dm_os_sys_memory)

SET @Usage_memory = @tot_memory - @available_memory

SET @memory_usage =(SELECT (100 -( available_physical_memory_kb /( total_physical_memory_kb * 1.0 ))* 100 ) memory_usage
FROMsys.dm_os_sys_memory)

SET @CPU_Utilization =
(SELECTtop 1
(CONVERT(varchar(30),record_time,21)+','+CONVERT(varchar(20),SQLProcessUtilization)+','+CONVERT(varchar(20),SystemIdle)+','+CONVERT(varchar(20),(100 - SystemIdle - SQLProcessUtilization)))AS CPUUtilization
FROM (
SELECT
record.value('(./Record/@id)[1]','int')AS record_id,
record.value('(./Record/SchedulerMonitorEvent/SystemHealth/SystemIdle)[1]','int')AS SystemIdle,
record.value('(./Record/SchedulerMonitorEvent/
```

	Avg_CPU_Utilized_in_Seconds	CPU_Utilized_in_Seconds	Execution count	Individual Query	Parent Query	Last Execution Time
1	21.303217666666666	63.909653000	3	UPDATE [Emp_Master] SET Name=UT.Name,Email=UT.E...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:41.893
2	16.633618000000000	49.900854000	3	insert into Emp_Master_Main(empid,name,email,nt_id,status,...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:26:58.983
3	6.169352000000000	6.169352000	1	Delete from Emp_Master_Main where NT_ID = ''	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:18.603
4	5.624321666666666	16.872965000	3	UPDATE #UserTable SET IsEmpIdExist = 1 FROM [Emp_M...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:35.000
5	3.561870333333333	10.685611000	3	INSERT INTO #UserTable(EmpID,Name,Email,NTID,MgrID...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:30.807
6	3.360192000000000	10.080576000	3	Update Emp_Master_Main set [NT_ID] = REPLACE([NT_ID...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:24.520
7	2.934167333333333	8.802502000	3	UPDATE [Emp_Master] SET IsActive =0 -- Pr...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:28.220
8	0.886383666666666	2.659151000	3	Update [Emp_Master] set [Emp_Master].First_Name = tblRM...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:28:09.197
9	0.419024000000000	0.419024000	1	update Emp_Master_Main set [status]='Attrited' where...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:24.343
10	0.257681000000000	0.773043000	3	INSERT INTO [Emp_Master] (EmpID,Name,Email,NT_ID,St...	CREATE PROCEDURE [dbo].[SyncEDtoCDPv...	2015-04-15 08:27:41.597

Figure 4. CPU and memory utilization and availability.

```
SystemHealth/ProcessUtilization)[1]','int')AS
SQLProcessUtilization,
    record_time
    FROM (
    select
    dateadd(ms, r.[timestamp] -sys.ms_ticks,getdate())as
record_time,
    cast(r.record asxml) record
    fromsys.dm_os_ring_buffers r
    crossjoinsys.dm_os_sys_infosys
    where
    ring_buffer_type='RING_BUFFER_SCHEDULER_
MONITOR'
    AND
    record LIKE'%<SystemHealth>%'
    )AS x
    )AS y )
```

The results of the above query execution have been captured in the Figure 4. This SQL statement can be scheduled and these results can be stored into some physical table for future analysis. Doing this way, CPU and memory status will be captured on every millisecond and will make sure either CPU or memory is reaching maximum level, root cause of the actual issue and what time the server is undergoing performance issue.

From the above results, complete information about CPU and memory usage availability will be obtained. From the above Figure 4, it can be noted that

- Column "Event Time" gives the actual timing of the event occurs.
- Immediate next columns "Total Memory GB, Usage Memory GB, Available Memory GB, Memory Usage" give the detailed information about total system memory, memory used by all the process, memory availability and memory usage, respectively. First three columns results are measured by GB and last column is measured by percentage.
- Next three columns "CPU Other Process Utilization, CPU SQL Process Utilization, CPU Idle" give the complete details about CPU utilized by other process, CPU utilized by SQL server process and current CPU availability, respectively.

5. Conclusion

This paper gives a clear, concise, and actionable plan and that can be used to tame the wildest of SQL problems. With these techniques provided in this paper, along with practice and research, optimizing SQL performance is an attainable execution plan. Extensive experiments are conducted to estimate and enhance the performance of the proposed approaches. The approaches presented here can be customized according to the business situation and SQL performance problems can be fixed that have plagued for weeks/while.

6. References

1. Jadidinejad AH, Sadr H. Improving weak queries using local cluster analysis as a preliminary framework. Indian Journal of Science and Technology. 2015 Jul; 8(15). ISSN (Online): 0974-5645.
2. Nagarajan S, Chandrasekaran RM. Design and implementation of expert clinical system for diagnosing diabetes using data mining techniques. Indian Journal of Science and Technology. 2015 Apr; 8(8):771–6.
3. Fritchey G. SQL server 2012 query performance tuning. 3rd Edition. USA: Apress; 2012 Jun 12.
4. Lobel L, Brust A. Programming Microsoft SQL server 2012. Microsoft Press. 2012.
5. Strate J, Krueger T. Expert performance indexing for SQL server 2012. USA: Apress; 2012.
6. Jorgensen A, Segarra J, Leblanc P, Chinchilla J, Nelson A. Microsoft SQL server 2012 Bible, Indianapolis, Indiana, USA: Johns Wiley and Sons Inc; 2012.
7. Henderson K. Transact-SQL (Titlul original: The guru's guide to transact-SQL) Bucuresti, Romania: Editor Teora; 2002.
8. Dejan S. Microsoft SQL server 2005 stored procedure programming in T-SQL & .NET, Third Edition. McGraw-Hill; 2006.
9. Andrew NK, Robust C. Optimization for performance tuning of modern database systems. European Journal of Operational Research. 2006; 171(2):412–29.
10. Kim SW, Jeong BS. Performance bottleneck of subsequence matching in time-series databases: Observation, solution, and performance evaluation. Information Sciences. 2007; 177(22):4841–58.
11. Pinal D. SQL SERVER – Optimization rules of thumb – best practices. Reader's Article; 2008 Apr 26.
12. KimW, David S, Batory RDS. Query processing in database systems. Springer Verlag. Berlin Heidelberg New York Tokyo; 2009.
13. Elmasri R, Navathe S B. Fundamentals of database systems. Fifth edition. Pearson Education; 2009.
14. Volkov A. SQL server optimization. 2015 May. Available from: http://msdn.microsoft.com/en-us/library/aa964133(v=sql.90).aspx
15. SQL tuning or SQL optimization. 2015 May. Available from: http://beginner-sql-tutorial.com/sql-query-tuning.htm
16. Han J, Kamber M. Data mining concepts and techniques. 2nd edition. Amsterdam, Netherlands: Elsevier Publisher; 2006. p. 383–5.

17. Han J, Kamber M. Data mining concepts and techniques. 2nd ed. Burlington, Massachusetts: Morgan Kaufmann; 2006. p. 285–8.
18. Rajesh K, Sangeetha V. Application of data mining methods and techniques for diabetes diagnosis. IJEIT. 2012; 2(3):224–9.
19. Ferreira D, Oliveira A, Freitas A. Applying data mining techniques to improve diagnoses in neonatal jaundice. BMC Med Informat Decis Making. 2012; 12:43.