IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

Performance Testing and Monitoring SQL Queries for Rebuild or Reorganize Operations

Mr. Sudhakar Panigrahy[1], Mr. Pragnyaban Mishra[2], Mr. Murali Krishna Senapaty[3]

Dept of CSE, Gandhi Institute of Engineering & Technology, Gunupur, Rayagada, India[1, 2, 3]

Abstract: A database system must be able to respond to requests for information from the user—i.e. process queries. Obtaining the desired information from a database system in a predictable and reliable fashion is Query Processing. To extract these results in a timely manner is possible with the technique of Query Optimization. For effective query processing the data must not be much fragmented. In this paper we focussed on the poor performance of query when the date fragmentation percentage is more and we have analysed how the fragmentation can be reduced using rebuild and re-organize techniques. In the last section, we have taken a sample data for query processing before rebuild and after that to have a comparative analysis.

Keywords: Indexing, Fragmentation, rebuild, re-organize.

1. INTRODUCTION

1.1 Index :

Index is used to speed up the query retrieval process. Index uses lookup table the search engine of the database uses to improve the performance of select statement of SQL query. As index stores data in sorted order hence it is also useful in order by clause. However index reduces the DML (Data Manipulation Language) operations (Insert, Update and Delete operations). Hence index is avoided in following situations:

- If number of rows in table is less(less than 1000).
- If the column not used in where clause of SQL statement.
- The number of distinct values of index column is less(Like the gender column contains M/F).
- The column contains more number of NULL values.
- If the column that is frequently updated.

Use of indexes is one of the best ways to improve performance of database application. The index uses the data structure B-Tree for indexes. The searching operation is first as B Tree uses logarithmic time for look up, insertion and deletion. The correct use of index requires careful analysis, benchmarking and testing. SQL query tuning is used to identify the useful of index[2].

The performance of the query is very important when the database size if large so there exist so many techniques for tuning the query by which the performance of the query can be improved.[1]

Now it is important to know, if we are looking for a specific value how it retrieves other values. Index also stores pointer to the corresponding rows in the table. Hence it retrieves data as per specific column value used where clause.

1.2 FRAGMENTATION:

Storing of data non-contiguously on a disk is called as fragmentation.

We can classify fragmentation into two types:

Types of Fragmentation

- **Internal Fragmentation**: When records are stored non-consecutively inside the page, then it is called internal fragmentation. Internal fragmentation will occur if there is unused space between records in a page. This fragmentation occurs sue to the process of data modifications (INSERT, UPDATE, and DELETE statements) that are made against the table many times and also to the indexes defined on the table. When the modifications are not equally distributed among the rows of the table and indexes then the fullness of each page can vary. Due to these unused spaces it causes poor cache utilization and more I/O, which finally leads to poor query performance.
- **External Fragmentation**: When the extents of a table are not physically stored contiguously on disk, switching from one extent to another causes higher disk rotations, and this is called Extent Fragmentation.
- **Logical Fragmentation**: Index pages also maintain a logical order of pages inside the extent. Every index page is linked with previous and next page in the logical order of column data. However, because of Page Split, the pages turn into out-of-order pages. An out-of-order page is a page for which the next physical page allocated to the index is not the page pointed to by the next-page pointer in the current leaf page. This is called Logical Fragmentation.

Advantages of Fragmentation

- **Usage:** Generally, applications work with views more than entire relations. That is why, for data distribution, it seems appropriate to work with subsets of relation as the unit of distribution.
- **Efficiency:** Data is stored close to where it is most frequently used. The data that is not needed by local applications is not needed to store locally.

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

- **Parallelism:** With the fragments as the unit of distribution, a transaction can be divided into several sub queries that operate on fragments. This allows concurrency, thereby allowing transactions that can do so safely to execute in parallel.

Disadvantages of the Fragmentation:

- **Performance:** The performance of global application that requires data from several fragments located at different sites may be slower.
- **Integrity:** Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.

2. REORGANIZE AND REBUILD INDEXES

The Database Engine maintains indexes whenever insert, update, or delete operations are made to the underlying data. Over time these modifications causes the information in the index to become scattered in the database. Fragmentation exists when indexes have pages in which the logical ordering, based on the key value, does not match the physical ordering inside the data file. More fragmented indexes can degrade query performance and cause your application to respond slowly[5].
We can remedy index fragmentation by reorganizing or rebuilding an index. For partitioned indexes built on a partition scheme, you can use either of these methods on a complete index or a single partition of an index.

2.1 Rebuild: Rebuilding an index **drops and re-creates** the index. This removes fragmentation, reclaims disk space by compacting the pages based on the specified or existing fill factor setting, and then reorders the index rows in contiguous pages. When ALL is specified, all indexes on the table are dropped and rebuilt in a single transaction.

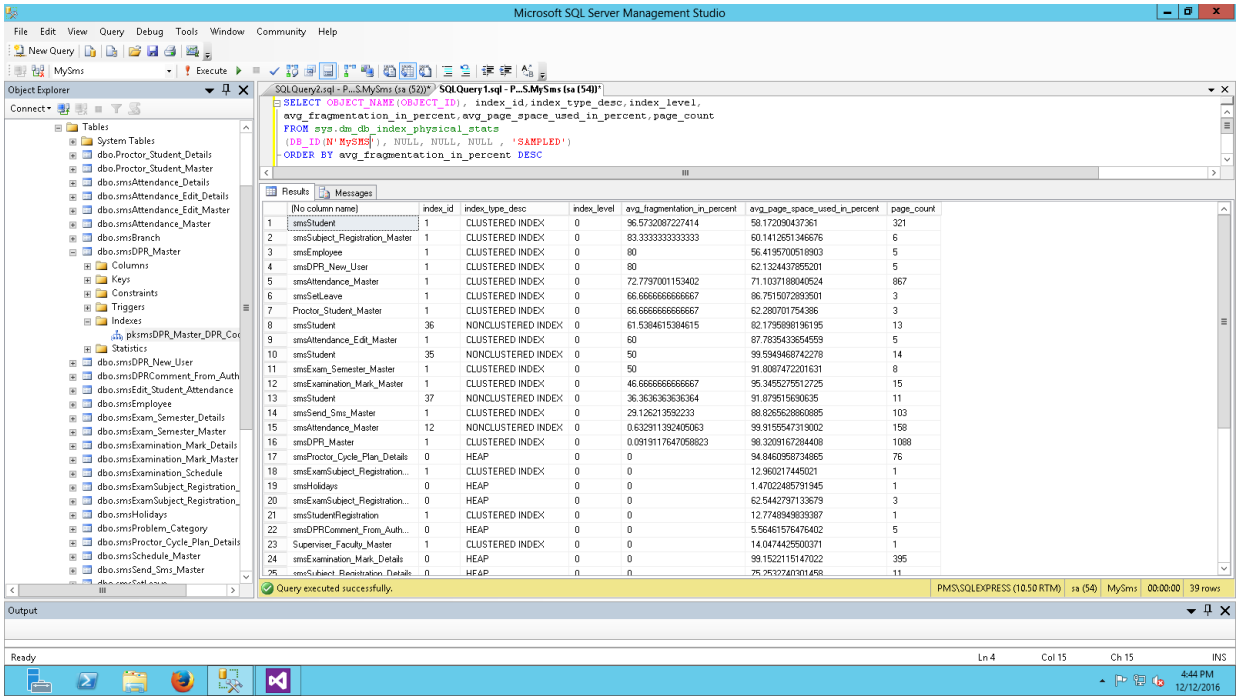
2.2 Reorganize: Reorganizing an index uses minimal system resources. It defragments the leaf level of clustered and non-clustered indexes on tables and views by physically reordering the leaf-level pages to match the logical, left to right order of the leaf nodes. Reorganizing also minimizes the index pages. The Compaction is also based on the existing fill factor value.

3. EXPERIENTIAL SETUP

3.1 DETECTING FRAGMENTATION

The first step in deciding which defragmentation method to use is to analyse the index to determine the degree of fragmentation. By using the system function sys.dm_db_index_physical_stats, we can detect fragmentation in a specific index, all indexes on a table or indexed view, all indexes in a database, or all indexes in all databases [4].

Query-1: The following query is used to detect the fragmentation exist in all indices of specific database
SELECT OBJECT_NAME(OBJECT_ID), index_id,index_type_desc,index_level,avg_fragmentation_in_percent,avg_page_space_used_in_percent,page_count FROM sys.dm_db_index_physical_stats (DB_ID(N'MySms'), NULL, NULL, NULL , 'SAMPLED') ORDER BY avg_fragmentation_in_percent DESC
Output
Fragmentation Before rebuild:



(Figure-3.1)

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

From the above output it is observed that the fragmentation exist in many indices of the database.

TABLE-3.1

Column	Description
avg_fragmentation_in_percent	The percent of logical fragmentation (out-of-order pages in the index).
fragment_count	The number of fragments (physically consecutive leaf pages) in the index.
avg_fragment_size_in_pages	**Average number of pages in one fragment in an index.**

3.2 Constructing complex query:
For measuring the performance of the query we used following tables:
1. smsStudent(no. Of records 15000),
2. smsAttendance_Master(no. Of records 118023),
3. smsAttendance_Details(no. Of records 6504012),
4. Proctor_Student_Master(no. Of records 1723),
5. Proctor_Student_Details(no of records 65230),
6. smsEmployee(no.of records 792)

From the above tables is has been seen that the larger tables are: smsAttendance_Master, smsAttendance_Details

Query-2: The query is used to find the attendance percentage of the students in specified date:

```
select smsStudent.vcRoll_No AS 'Roll No',smsStudent.vcStudent_Name as Name,smsStudent.vcStudent_Mobile_No
as 'Student Mobile No',
vcFathers_No as 'Fathers Mobile No',smsEmployee.vcEmp_Name as
'Proctor',count(smsAttendance_Master.vcAttendance_No) as 'Tot Classes',
SUM(intpresent) as Presnt,count(smsAttendance_Master.vcAttendance_No)-SUM(intpresent) as Absent ,
cast((cast(SUM(intPresent)*100 as decimal(18,2))/COUNT(smsAttendance_Master.vcAttendance_No))as
decimal(10,2)) as Atte_Per
from
smsStudent,smsAttendance_Master,smsAttendance_Details,Proctor_Student_Master,Proctor_Student_Details,smsEmp
loyee
where smsAttendance_Master.vcAttendance_No=smsAttendance_Details.vcAttendance_No and
smsAttendance_Details.vcRoll_No=smsStudent.vcRoll_No and
Proctor_Student_Master.vcProctor_Student_Code=Proctor_Student_Details.vcProctor_Student_Code and
Proctor_Student_Details.vcStudent_Roll_No=smsStudent.vcRoll_No and
Proctor_Student_Master.vcProctor_Employee_Code=smsEmployee.vcEmp_code and
smsStudent.vcBranch_Id='CSE' AND smsStudent.intSemester=6 and vcSection='A' and
smsStudent.vcStatus='ACTIVE' AND
smsAttendance_Master.vcStatus='ACTIVE'   and  smsAttendance_Master.dtDtAttendance_Date>='12/13/2016
12:00:00 AM' and
smsAttendance_Master.dtDtAttendance_Date<='12/13/2016 12:00:00 AM'
group by smsStudent.vcRoll_No,smsStudent.vcStudent_Name,smsStudent.vcStudent_Mobile_No,
vcFathers_No, smsEmployee.vcEmp_Name
having count(smsAttendance_Master.vcAttendance_No)-SUM(intpresent)>=1 order by smsStudent.vcRoll_No
```

Query-3: The query is used to rebuild all indices of the database.

```
use MySms;// MySms is name of the database
SET NOCOUNT ON;
DECLARE @objectid int;
DECLARE @indexid int;
DECLARE @partitioncount bigint;
DECLARE @schemaname nvarchar(130);
DECLARE @objectname nvarchar(130);
DECLARE @indexname nvarchar(130);
DECLARE @partitionnum bigint;
DECLARE @partitions bigint;
DECLARE @frag float;
DECLARE @command nvarchar(4000);
```

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

```
-- Conditionally select tables and indexes from the sys.dm_db_index_physical_stats function
-- and convert object and index IDs to names.
SELECT
   object_id AS objectid,
   index_id AS indexid,
   partition_number AS partitionnum,
   avg_fragmentation_in_percent AS frag
INTO #work_to_do
FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL , NULL, 'LIMITED')
WHERE avg_fragmentation_in_percent > 10.0 AND index_id > 0;
-- Declare the cursor for the list of partitions to be processed.
DECLARE partitions CURSOR FOR SELECT * FROM #work_to_do;
-- Open the cursor.
OPEN partitions;
-- Loop through the partitions.
WHILE (1=1)
   BEGIN;
      FETCH NEXT
        FROM partitions
        INTO @objectid, @indexid, @partitionnum, @frag;
      IF @@FETCH_STATUS < 0 BREAK;
      SELECT @objectname = QUOTENAME(o.name), @schemaname = QUOTENAME(s.name)
      FROM sys.objects AS o
      JOIN sys.schemas as s ON s.schema_id = o.schema_id
      WHERE o.object_id = @objectid;
      SELECT @indexname = QUOTENAME(name)
      FROM sys.indexes
      WHERE  object_id = @objectid AND index_id = @indexid;
      SELECT @partitioncount = count (*)
      FROM sys.partitions
      WHERE object_id = @objectid AND index_id = @indexid;
-- 30 is an arbitrary decision point at which to switch between reorganizing and rebuilding.
      IF @frag < 30.0
         SET @command = N'ALTER INDEX ' + @indexname + N' ON ' + @schemaname + N'.' + @objectname + N'
REORGANIZE';
      IF @frag >= 30.0
         SET @command = N'ALTER INDEX ' + @indexname + N' ON ' + @schemaname + N'.' + @objectname + N'
REBUILD';
      IF @partitioncount > 1
         SET @command = @command + N' PARTITION=' + CAST(@partitionnum AS nvarchar(10));
      EXEC (@command);
      PRINT N'Executed: ' + @command;
   END;

-- Close and deallocate the cursor.
CLOSE partitions;
DEALLOCATE partitions;
-- Drop the temporary table.
DROP TABLE #work_to_do;
GO
```

Output:

```
Executed: ALTER INDEX [PK__smsEmplo__99AAB30420C1E124] ON [dbo].[smsEmployee] REBUILD
Executed: ALTER INDEX [IX_smsStudent_Branch] ON [dbo].[smsStudent] REORGANIZE
Executed: ALTER INDEX [IX_smsStudent_Semester] ON [dbo].[smsStudent] REORGANIZE
Executed: ALTER INDEX [IX_smsStudent_Section] ON [dbo].[smsStudent] REBUILD
```

(Note : All rows of the output are not shown here)

From the study [3] it is noticed that rebuild is not useful when the fragmentation is less than 30%. So the query is designed to reorganize or rebuild as per the fragmentation percentage as below:

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

TABLE-3.2

avg_fragmentation_in_percent value	Corrective statement
> 5% and < = 30%	ALTER INDEX REORGANIZE
> 30%	ALTER INDEX REBUILD WITH (ONLINE = ON)*

Fragmentation after rebuild all indices (Execution of the Query-3)

TABLE-3.3

(No column name)	index_ id	index_type_desc	index_ level	avg_ fragmentation _in_percent	avg_page_ space_used _in_percent	page_ count
smsEmployee	1	CLUSTERED INDEX	0	66.66666667	94.0490981	3
Proctor_Student_Details	7	NONCLUSTERED INDEX	0	50	93.74845565	12
smsStudent	36	NONCLUSTERED INDEX	0	45.45454545	97.1258216	11
smsStudent	35	NONCLUSTERED INDEX	0	35.71428571	99.59494687	14
smsStudent	37	NONCLUSTERED INDEX	0	27.27272727	91.87951569	11
smsSend_Sms_Master	1	CLUSTERED INDEX	0	4.301075269	98.38045466	93
smsAttendance_Master	12	NONCLUSTERED INDEX	0	0.625	99.41423277	160
smsAttendance_Master	1	CLUSTERED INDEX	0	0.319488818	99.14720781	626
smsDPR_Master	1	CLUSTERED INDEX	0	0.091911765	98.34676303	1088
smsProctor_Cycle_Plan _Details	0	HEAP	0	0	94.84609587	76
smsExamSubject_ Registration_Master	1	CLUSTERED INDEX	0	0	12.96021745	1
smsHolidays	0	HEAP	0	0	1.470224858	1
smsExamSubject_ Registration_Details	0	HEAP	0	0	62.54427971	3
smsStudentRegistration	1	CLUSTERED INDEX	0	0	13.24437855	1
smsDPRComment_ From_Authorites	0	HEAP	0	0	5.564615765	5
smsSetLeave	1	CLUSTERED INDEX	0	0	86.75150729	3
Superviser_Faculty _Master	1	CLUSTERED INDEX	0	0	14.05979738	1
smsSubject_ Registration _Master	1	CLUSTERED INDEX	0	0	90.22424018	4
smsExamination_Mark _Details	0	HEAP	0	0	99.15221151	395
smsSubject_ Registration_Details	0	HEAP	0	0	75.20610329	11
smsExam_Semester_Master	1	CLUSTERED INDEX	0	0	89.28836175	8
smsAttendance_Details	0	HEAP	0	0	98.7805535	10100
smsBranch	1	CLUSTERED INDEX	0	0	9.340252039	1
smsExam_Semester_Details	0	HEAP	0	0	95.30226093	68

IJARCCE

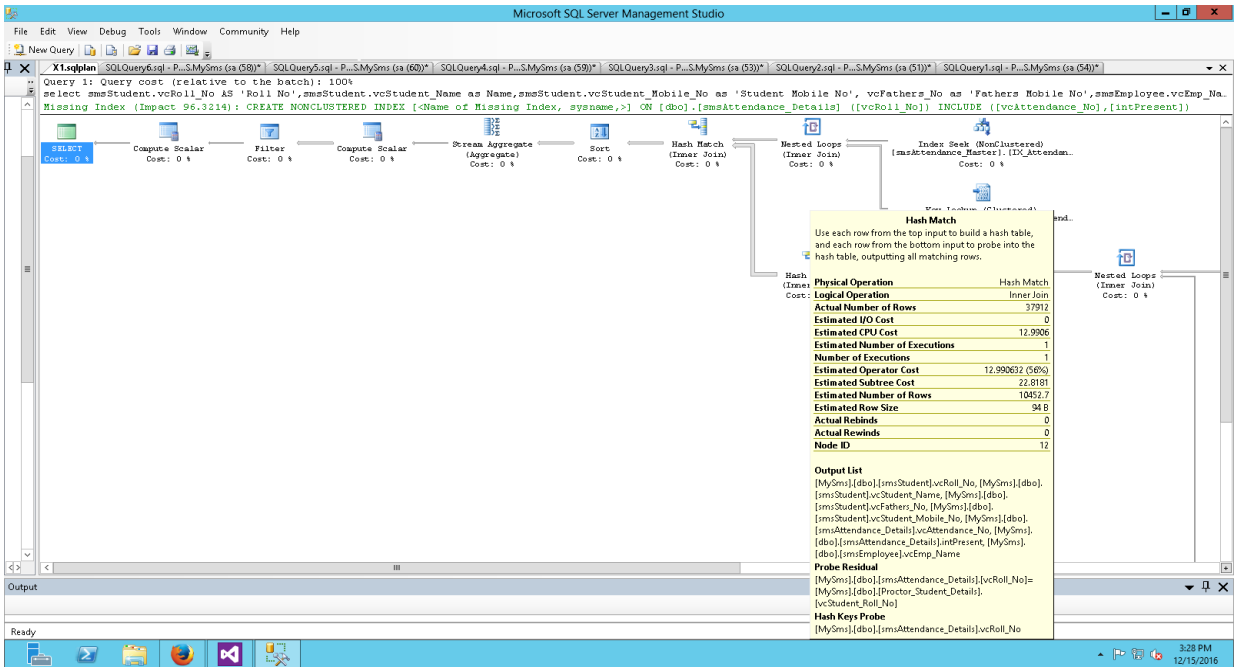
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

smsUploadedStudent Imagedetails	1	CLUSTERED INDEX	0	0	0.852483321	1
smsUploadedStudent Imagedetails	1	CLUSTERED INDEX	0	0	0	0
smsStudentRegistration Temp	0	HEAP	0	0	94.19570052	40
Proctor_Student_Master	1	CLUSTERED INDEX	0	0	93.43958488	2
sysdiagrams	1	CLUSTERED INDEX	0	0	0	0
sysdiagrams	2	NONCLUSTERED INDEX	0	0	0	0
Proctor_Student_Details	0	HEAP	0	0	78.41057574	29
smsSchedule_Master	1	CLUSTERED INDEX	0	0	10.05683222	1
smsEdit_Student_ Attendance	1	CLUSTERED INDEX	0	0	0	0
smsAttendance_Edit _Master	1	CLUSTERED INDEX	0	0	88.36916234	5
smsAttendance_Edit _Details	0	HEAP	0	0	50.86278725	12
smsProblem_Category	1	CLUSTERED INDEX	0	0	1.890289103	1
smsExamination_Schedule	1	CLUSTERED INDEX	0	0	8.549542871	1
smsExamination_Mark _Master	1	CLUSTERED INDEX	0	0	95.34552755	15
smsDPR_New_User	1	CLUSTERED INDEX	0	0	77.6933531	4
smsStudent	1	CLUSTERED INDEX	0	0	97.80305164	191

4. PERFORMANCE MEASUREMENT OF REBUILD OR REORGANIZE OPERATION

Execution plan is as below



IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

4.1 Tools identified to measure the performance:

The following tools are used to trace the performance result and necessary actions:

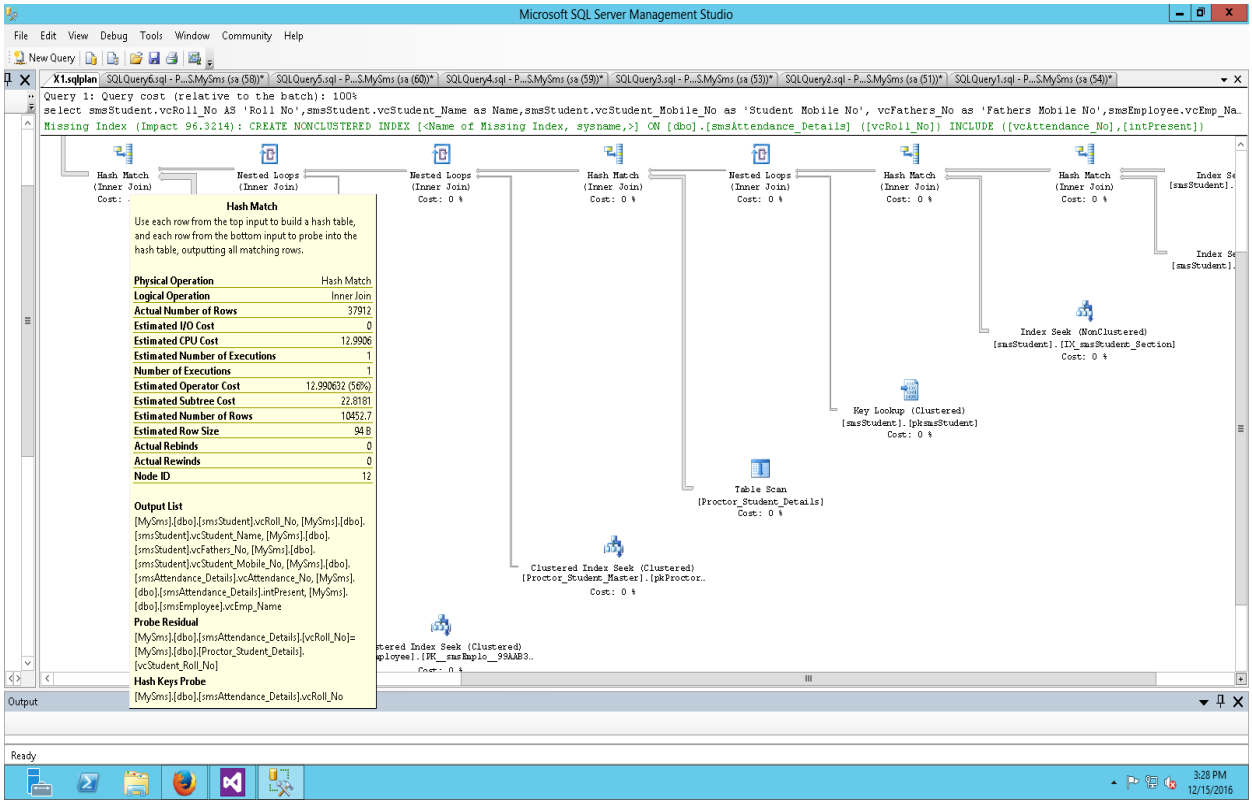
4.1.1: Estimated Execution Plan

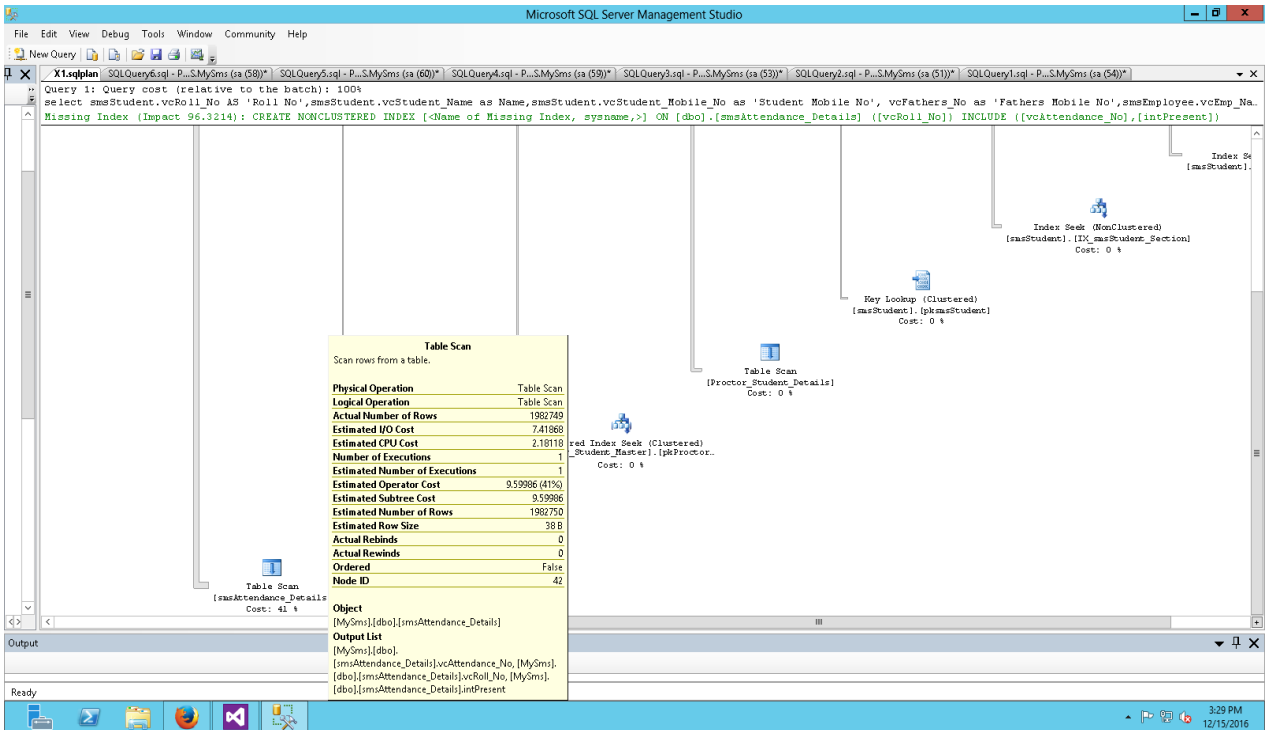
4.1.2: Client Statistics

4.1.3: Measurement of execution time using query

4.1.1: Estimated Execution Plan:

It displays the execution plan, resources (execution time and space) of the current query and also it necessary actions require to improve the performance of the query. Following is the output of the execution plan after rebuild or reorganize operations (execution of the query-3)





IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

4.1.2: Client Statistics

The client statistics report displays the performance result in different trials about Query Profile Statistics, Number of INSERT, DELETE and UPDATE statements, Rows affected by INSERT, DELETE, or UPDATE statements, Number of SELECT statements , Rows returned by SELECT statements, Number of transactions , Network Statistics, Number of server roundtrips, TDS packets sent from client, TDS packets received from server, Bytes sent from client, Bytes received from server, Time Statistics, Client processing time, Total execution time, Wait time on server replies.

The output is as below:

Results | Messages | Client Statistics

	Trial 4	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	19:21:34	19:21:31	19:21:28	19:16:17	
Query Profile Statistics					
Number of INSERT, DELETE and UPDATE statements	0 →	0 →	0 →	0 →	0.0000
Rows affected by INSERT, DELETE, or UPDATE statem...	0 →	0 →	0 →	0 →	0.0000
Number of SELECT statements	1 →	1 →	1 →	1 →	1.0000
Rows returned by SELECT statements	12 →	12 →	12 →	12 →	12.0000
Number of transactions	0 →	0 →	0 →	0 →	0.0000
Network Statistics					
Number of server roundtrips	1 →	1 →	1 →	1 →	1.0000
TDS packets sent from client	1 →	1 →	1 →	1 →	1.0000
TDS packets received from server	1 →	1 →	1 →	1 →	1.0000
Bytes sent from client	3112 →	3112 →	3112 →	3112 →	3112.0000
Bytes received from server	1468 →	1468 →	1468 →	1468 →	1468.0000
Time Statistics					
Client processing time	0 →	0 →	0 →	0 →	0.0000
Total execution time	671 ↓	687 ↓	703 ↓	1234 →	823.7500
Wait time on server replies	671 ↓	687 ↓	703 ↓	1234 →	823.7500

4.1.3: Measurement of execution time using query

```
DECLARE @EndTime datetime
DECLARE @StartTime datetime
SELECT @StartTime=GETDATE()

        /* The Query 2 need paste here to measure the excution time*/
SELECT @EndTime=GETDATE()

--This will return execution time of your query
SELECTDATEDIFF(ms,@StartTime,@EndTime)AS [Duration in millisecs]
```

Output is as below: 60 rows are retrieved

```
DECLARE @EndTime datetime
DECLARE @StartTime datetime
```

Results | Messages

	Roll No	Name	Student Mobile No	Fathers Mobile No	Proctor	Tot Classes	Presnt	Absent	Atte_Per
1	14CSE006	DHANANJAY KUMAR	7064102942	8409541403	Mr.Ranjeet Panigrahi	2	0	2	0.00
2	14CSE021	ANKIT KUMAR	9556593946	9905949176	Mrs.Gitanjali Mishra	2	0	2	0.00
3	14CSE028	SURAJ KUMAR SANDHA	7205676499	9777405365	Mrs.Gitanjali Mishra	2	0	2	0.00
4	14CSE043	KUMAR SARTHAK	8018995900	9437270658	Dr. Nilambar Sethy	2	0	2	0.00
5	14CSE051	NISHANT DUTTA	7538919448	9334807912	Mr. Amiya Ku . Sahu	2	0	2	0.00
6	14CSE062	DIMMALA KUMAR SWAMY	7205372382	9777978982	Mr. Amiya Ku . Sahu	2	0	2	0.00
7	14CSE071	NIRANJAN EKKA	9178536184	8658718105	Mr. Amiya Ku . Sahu	2	0	2	0.00
8	14CSE074	SUBHASREE CHOUDHURY	8093966015	9437722832	Mr. Amiya Ku . Sahu	2	0	2	0.00
9	14CSE078	D PRATEEK DORA	9556111853,8895311827	9437058827	Mr Sudhakar Panigrahy	2	0	2	0.00

	Duration in millisecs
1	696

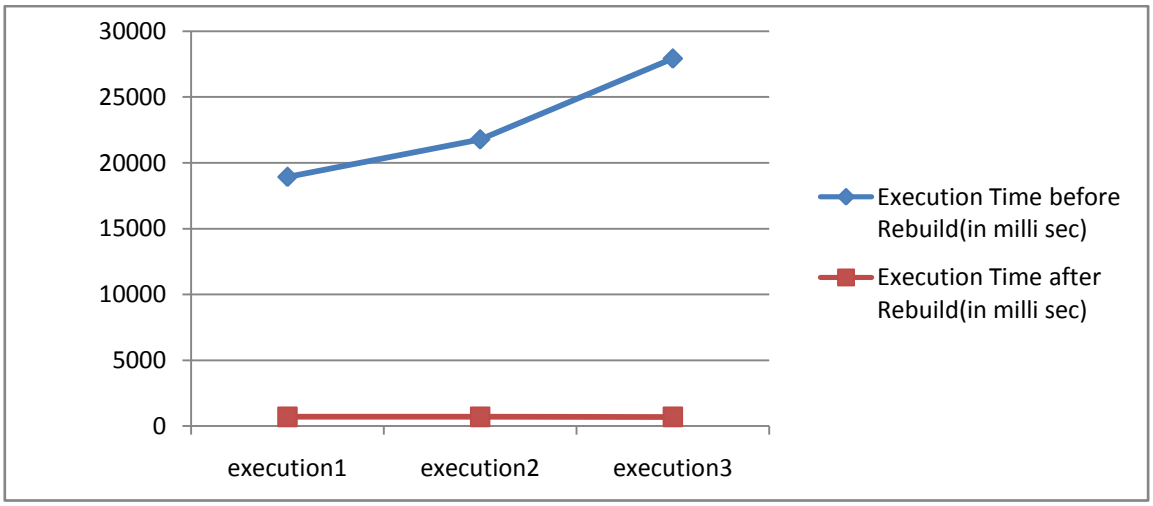
IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

5. RESULT ANALYSIS

5.1: Execution Time:

Following table represents the execution time in milli seconds different executions. The execution time is measured before and after the rebuild operations:

	execution1	execution2	execution3
Execution Time before Rebuild(in milli sec)	18932	21787	27933
Execution Time after Rebuild(in milli sec)	696	699	690



5.2: Fragmentation percentage comparison:

After rebuild and reorganize the fragmentation level also reduced refer the table-4.3 and figure-4.1

It is observed that the fragmentation of different tables whose page size is more, the fragmentation percentage is reduced about to 0% (table name : smsAttendance_master, the fragmentation of the non cluster index for date field is reduced to 0.4632%)

From the tables 4.3 it is observed that after rebuild or reorganize of indices the fragmentation still exist in the tables: smsStudent, smsEmployee. The reason for the fragmentation is the page size of the specific tables is less. Hence there is no distinct improvement of fragmentation of the specified tables.[6]

6. CONCLUSION

One of the most important functional requirements of a database system is its ability to process queries in a timely manner. This is particularly true for very large database applications. There are different types of indices are there like cluster index and non-cluster index. The performance of the query decreases after couple of DML statements like insert, update, delete which increases the fragmentation. So in experiment it is observed that the performance of the query decreases drastically due to the fragmentation. Hence rebuild and reorganize have to be performed periodically after identifying the fragmentations. The rebuilt is necessary when the fragmentation is more than 30% and reorganize is necessary when the fragmentation is <30%. The rebuild operations takes the database to offline mode where as the reorganize operation reduces the fragmentation in online mode operations. Thus, a great deal of research and resources is spent on creating smarter, highly efficient query optimization engines. The rebuild is simplest and frequent used method for optimized query processing.

REFERENCES

[1] "Performance Tuning in Microsoft sql Server DBMS" by Sapna Dahiya, Pooja Ahlawat, IJCSMC, Vol. 4, Issue. 6, June 2015, pg.381 – 386

[2] "Increasing Database Performance using Indexes", by Cecilia CIOLOCA, Mihai GEORGESCU, ROMANIA, Database Systems Journal vol. II, Issue 2, 2011

[3] "Introduction to Query Processing and Optimization" Michael L. Rupley, Jr.Indiana University at South Bendmrupleyj@iusb.edu

[4] http://ecomputernotes.com/database-system/adv-database/fragmentation

[5] https://msdn.microsoft.com/en-in/library/ms189858.aspx

[6] "Introduction to Query Processing and Optimization, International Journal of Advanced Research in Computer Science and Software Engineering" Vol. III, Issue 7, July 2013,