Optimizing SQL Performance in a Parallel Processing DBMS Architecture

[1]Nayem Rahman and [2]Leonard Sutton
[1]Cross Enterprise Systems IT, Intel Corporation, Hillsboro, OR, USA,
Email: nayem.rahman@intel.com
[2]Independent Teradata Consultant, The Boeing Company, Seattle, WA, USA,
LeonardSuttonLLC@bctonline.com

Abstract

A Database Management System (DBMS) with a parallel processing architecture is different from conventional database systems. Accordingly, writing SQL for a parallel processing DBMS architecture requires special attention to maintain parallel efficiency in DBMS resources usage such as CPU and I/O. In a large data warehouse, a large number of SQL queries are executed by different user groups on a daily basis. Query response time needs to be minimal. Many batch jobs run to refresh data warehouse subject areas several times a day. To allow batch cycles run more frequently and keep the data warehouse environment stable the database system's resource utilization must be optimal. Running efficient queries is critical to keep resource utilization manageable. This article discusses the techniques of SQL writing, tuning, utilization of index, data distribution techniques in a parallel processing DBMS architecture. We hope that these techniques will empower SQL developers and business intelligence community to write efficient queries which will help maintain a stable data warehousing environment.

Keywords—Database Architecture, DBMS, Computing Resources, CPU, I/O, Data Warehouse, Parallel Processing

An earlier version, "SQL Optimization in a Parallel Processing Database System" byRahman, N. appeared in Proceedings of CCECE '13. ©2013 IEEE.

© University of Liberal Arts Bangladesh
All rights reserved.

Manuscript received on 29 August 2016 and accepted for publication on 17 September 2016.

1 INTRODUCTION

TODAY'Sbusiness organizations use data warehouse as a central repository of data that come from internal operational sources as well as external sources (includes big data) [15]. As business organizations become global, there is a need to run business operations twenty four-by-seven so business decisions could be made faster. The data warehouse plays a prominent role in providing business intelligence (BI) capabilities [46]. It has proved to be one of the key infrastructures of information technology for an organization to better manage and leverage its information [31, 45]. Data warehouses are used for target marketing, financial reporting, customer services, inventory management, and more. They keep changing the way business is conducted [9]. Research suggests that data warehouses are increasingly being used by medium and large companies as these organizations are realizing its benefits [36].

Due to global nature of business and increased competition the data warehouse users and analytical community want to get near real-time information for strategic and tactical decision making. With the increased capabilities of advanced database technologies and massive parallel processing systems, it is now possible to load, maintain, and access databases of terabyte size [14] in reasonable times. In order to maintain a stable data warehousing environment data warehouse design, SQL writing, and load techniques all need to be efficient [2, 27]. Strategies are needed to save database management system (DBMS) resources during load processes in order to make the DBMS available to analytical tools and query processing while data warehouse refreshes continue at the same time. The data warehouse SQL queries for both load process and reporting need to be efficient. In this article we propose a comprehensive list of SQL query optimization techniques. We argue that data warehouse resource consumption could be made optimal by taking advantage of parallel processing architecture of database system.

This article is organized as follows: Section 2 briefly discusses related work done in this area. Section 3 discusses our proposed techniques of performance optimization. Section 4 discusses SQL parallel efficiency implementation steps and DBMS resource savings. Section 5 provides SQL query performance metrics of use cases. Section 6 summarizes and concludes the article.

2 LITERATURE REVIEW

Researches have been conducted in different area of data warehousing. These include design issues [11, 13, 17, and 19], extract-transform-load (ETL) tools [23, 43], temporal data updates [18], data warehouse automation [24], data maintenance [1], implementation issues [21] and implementation effectiveness [36]. In this paper, we attempt to address the question of how to make a data warehousing environment stable and how to keep resource consumption by individual queries optimal by virtue of efficient SQL writing.

In data warehousing and data management systems parallel processing architecture is considered as a key capability [10]. Database system performance and SQL query optimization are important in any database system [39, 47]. In real world, the SQL queries that get executed are often quite complex and for data mining tasks queries are even more complex and resource intensive [38]. Hence, SQL query optimization is very critical for data warehouse stability.

In order maintain a stable data warehouse system in terms of resource utilization researchers and industry technical leaders propose many tools, techniques, algorithms and strategies. Here we take a cursory look at them. Ghazal et al. [20] present an algorithm that dynamically chooses between saving and re-using compiled plans and minimize re-compiling queries. Ganguly et al. [16] show that a cost model can predict response time with features of query execution parallelism. Kashem et al. [25] present a query optimization algorithm in rank aware queries to efficiently answer to the queries with join of N relations. Rahman and Rutz [24] assert it is critical to ensure that processes in the data warehouse are automated and optimized for performance. The authors propose using automation tools in a data warehouse ETL process, SQL block generations for views, stored procedures and macros wherever possible.

Elnaffar et al. [12] state that a DBMS workload could be considered as a determinant of performance tuning techniques. The authors argue that DBMS workloads are different in terms of OLTP and DSS. They propose reconfiguring DBMS resources by automatically identifying the DBMS work load. DSS queries process huge volume of data. Hence, they take more resources than OLTP queries. Dayal et al. [7] and Sharaf and Chrysanthis [41] propose managing database workloads with mixture of OLTP-like queries that run for fraction of a second and on the other hand, business intelligence queries that run for a longer time. The standard benchmark for Decision Support Systems comprises database workload and query performance metrics [42]. Powley et al. [40] present query throttling techniques as method to control workload. Kerkad et al. [26] propose a query beehive algorithm for data warehouse buffer management and query scheduling to improve data warehouse system's performance and scalability. Rahman [9] proposes a balanced scorecard approach for measuring performance of data warehouse operations.

Meng et al. [34] propose logically splitting large queries so each of them deal with small set of data and cause less impact on the overall warehouse environment and thus avoid consuming huge resource by one single large query. Narasayya et al. [35] propose a buffer pool page replacement algorithm that effectively shares buffer pool memory in multi-tenant relational database-as-a-service (DaaS). VanderMeer et al. [44] propose a cost-based database request across a cluster of databases to spread workload and resource usage. Neumann [37] asserts that query optimizer needs to be more efficient to efficiently handle different types of SQL queries. The author argues that query optimizer has larger impact than that of runtime system.

Hill and Ross [22] present a method to make outer joins efficient in order to improve query performance and response time. Rahman [18] discusses performance improvement of load and report queries, and maintenance of views with temporal data. Armstrong [4] proposes reduction of data movement to increase user accessibility, minimize data latency and improve performance of the entire data warehouse. Krompass et al. [28, 29] propose a workload management system for managing the execution of individual queries based on customer service level objectives. Rahman [49] proposes strict governance in data warehouse maintenance and operations to bring discipline and control. This includes defining guidelines for application developers and IT integration engineers to follow. The author presents a set of data warehouse governance best practices with insights from real-world experience and research findings from industry and academic papers.

Allen and Parsons [3] demonstrate that anchoring and adjustment during query reuse by novice query writers can lead to queries that are less accurate than those written from scratch. This suggests that in real-world SQL queries could be written by users and developers of varieties skill-set. A significant number of them could be badly written. Hence, SQL queries need to run through some SQL score-card process [1] to ensure parallelism of query runs. Lee et al. [30] propose a Statistical Process Control (SPC) charts to detect database performance anomalies and identify their root causes. However, performance anomalies could be prevented from happening if each SQL queries could be run through a SQL performance scorecard process [1].

In this article, we focus on writing efficient SQL that conforms to parallel processing architecture. We address the problem of DBMS resource consumption and stability issue by taking care of SQL efficiency, defining indexes and many other SQL optimization techniques. By taking advantage of database parallelism architecture the problem of SQL query response time could be minimized [8]. This helps in achieving database system resources (CPU and I/O) saving [5].

3 Performance Optimization in a Parallel Processing DBMS

In a parallel processing DBMS architecture a large number of individual Access Module Processors (AMP) are used. We can think of these as "Units of Parallelism". Each "unit" will have dedicated Disk and dedicated CPU. The goal of the Physical Database Design, and the design of the SQL submitted, is to force the processing to be as well distributed across all the AMPs as possible. Because the CPU and other resources are shared with other jobs across the system, the actual impact of any given process is: the highest amount of resources used on any one AMP, times the number of AMPs on that particular system. If the high AMP uses 80 CPU and 2000 I/O, and we have a 100 AMP system, then the real impact of that job is as if it used 8000 CPU and 20,000 I/O, even if the total resources used by all the AMPs appears to be a much smaller amount. When a query executes, each step in the process waits until all the AMPS are finished for a given step, before the next step starts. For this reason, the most efficient processes are the ones which have about the same amount of resources used on each AMP.

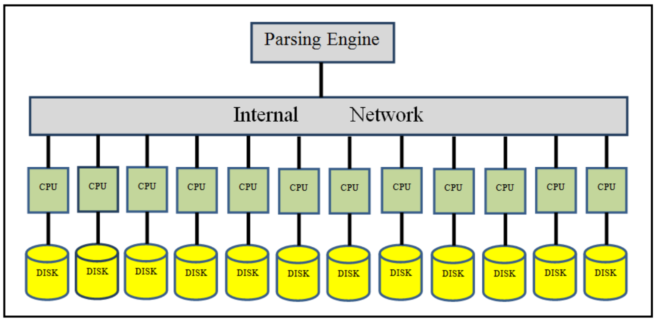

Figure 1: A Parallel Processing DBMS Architecture.

Skewed processing is when there is significant difference between the resources used by the "high" AMP and average of all AMP's. If the Physical Database Design has been verified to be optimizes, then attention can be given to the SQL being submitted. This document deals with different way to optimize the SQL.

3.1 Row Redistribution in a Parallel Processing Architecture

In most large systems, a typical report will need to look at many tables. In a parallel processing database system, care must be taken in choosing Primary and Secondary Indexes, to try to avoid "redistribution" steps in the SQL Parsing steps. The optimizer joins 2 tables at a time and puts the result into a spool file. Then it joins that to another table or spool file, and so on until all the tables are joined. On each of these joins the rows to be joined on each table must reside on the same AMP. If the 2 tables have the same primary index (PI) then all the rows that will join together already reside on the same AMP. If the 2 tables have different PI's the DBMS needs to do one of two things: either duplicate (one of the) table(s) on all AMPs or redistribute one of the tables (using a PI that is the same as the other table) so that the rows being joined now reside on the same AMP. So the reason for redistribution is always that the 2 tables being joined do not have the same PI. Sometimes we cannot do anything about this; it is just the way it works. Other times, we can build a derived table, narrowing the selection of rows to a smaller number, and try to make the optimizer duplicate the table on all AMPS. If it does not disturb other processes; the best way to eliminate redistribution is to build the tables being joined with the same PI (this is not always possible). Depending on the choice of indexes, this join process can have very different paths to get to the desired results.

3.2 Duplicating on all AMPs and Product JOINs

Sometimes, the optimizer sometimes builds a copy of a table on each unit of parallelism to facilitate parallel processing. There are many cases where this proves to be the best path for the optimizer to take. To ensure that this duplicating takes less resource, a derived table can be used in the SQL, creating a reduced set of rows and/or columns for the optimizer to work with.

Sometimes a Product-Joins occur when the optimizer needs to join a large and a small table. To improve performance: narrow down the rows and columns of that small table; if the smaller table contains static data with few records in that case column values could be placed in memory variables (Figure 2). That way, a JOIN with the smaller table could be entirely eliminated.

```
,CASE
        WHEN ANLC.fscl_yr_mo_nbr = :fscl_yr_mo_nbr THEN :last_day_curr_mo
        WHEN ANLC.fscl_yr_mo_nbr = :Mo1 THEN :Dt1
        WHEN ANLC.fscl_yr_mo_nbr = :Mo2 THEN :Dt2
        WHEN ANLC.fscl_yr_mo_nbr = :Mo3 THEN :Dt3
        WHEN ANLC.fscl_yr_mo_nbr = :Mo4 THEN :Dt4
        WHEN ANLC.fscl_yr_mo_nbr = :Mo5 THEN :Dt5
        WHEN ANLC.fscl_yr_mo_nbr = :Mo6 THEN :Dt6
        WHEN ANLC.fscl_yr_mo_nbr = :Mo7 THEN :Dt7
        WHEN ANLC.fscl_yr_mo_nbr = :Mo8 THEN :Dt8
        WHEN ANLC.fscl_yr_mo_nbr = :Mo9 THEN :Dt9
        WHEN ANLC.fscl_yr_mo_nbr = :Mo10 THEN :Dt10
        WHEN ANLC.fscl_yr_mo_nbr = :Mo11 THEN :Dt11
        WHEN ANLC.fscl_yr_mo_nbr = :Mo12 THEN :Dt12
        WHEN ANLC.fscl_yr_mo_nbr = :Mo13 THEN :Dt13
        WHEN ANLC.fscl_yr_mo_nbr = :Mo14 THEN :Dt14
        WHEN ANLC.fscl_yr_mo_nbr = :Mo15 THEN :Dt15
        WHEN ANLC.fscl_yr_mo_nbr = :Mo16 THEN :Dt16
        WHEN ANLC.fscl_yr_mo_nbr = :Mo17 THEN :Dt17
        WHEN ANLC.fscl_yr_mo_nbr = :Mo18 THEN :Dt18
        WHEN ANLC.fscl_yr_mo_nbr = :Mo19 THEN :Dt19
        WHEN ANLC.fscl_yr_mo_nbr = :Mo20 THEN :Dt20
        WHEN ANLC.fscl_yr_mo_nbr = :Mo21 THEN :Dt21
        WHEN ANLC.fscl_yr_mo_nbr = :Mo22 THEN :Dt22
        WHEN ANLC.fscl_yr_mo_nbr = :Mo23 THEN :Dt23
        WHEN ANLC.fscl_yr_mo_nbr = :Mo24 THEN :Dt24
        ELSE DATE '9999-12-31'
END AS depr_hist_clndr_dt
```

Figure2: Eliminate a skewed JOIN and populate column with memory variable values.

3.3 Parallel Efficiency

Skewed data distribution and skewed processing adversely affect parallel efficiency. Poor parallel efficiency occurs when the join field is highly skewed. Rows are redistributed to AMPs based on the join column values; a disproportionate number of rows may end up on one AMP on or a few AMPs operation. Highly non-unique PIs cause uneven row distribution. More than 1000 occurrences of a value in a Non-Unique Primary Index (NUPI) value begin to cause performance degradation problems: Increased I/O's for updates and inserts of over-represented values; Poor CPU parallel efficiency on full table scans and bulk inserts.

```
CREATE TABLE Capital_DRV_MET.fact_purch_doc_line
(
  asof_src_dt DATE NOT NULL,
  asof_src_ts TIMESTAMP(0) NOT NULL,
...
  src_sys_nm CHAR(20) CHARACTER SET LATIN NOT NULL,
  purch_doc_nbr CHAR(10) CHARACTER SET LATIN NOT NULL,
  purch_doc_line_nbr CHAR(5) CHARACTER SET LATIN NOT NULL,
  purch_doc_line_shrt_dsc VARCHAR(40) CHARACTER SET LATIN,
...
)
PRIMARY INDEX xfact_purch_doc_line02 (purch_doc_nbr,purch_doc_line_nbr);
UNIQUE INDEX xfact_purch_doc_line01 (purch_doc_nbr ,purch_doc_line_nbr,src_sys_nm );
```

Figure 3: Primary Index defined with two columns for better row distribution.

Figure 3 shows a Primary Index (PI) with two columns to make sure rows are distributed to all AMPs. Initially we defined index with a single column, that is, with 'purch_doc_nbr' only. But, since there are a large number of the same 'purch_doc_nbr' we redefined PI consisting of two columns. Addition of the second column, 'purch_doc_line_nbr' made data distribution much better. Table load performance has improved significantly. If there is still a need for an index on purch_doc_nbr, we can build a secondary index

3.4 Primary Index Choice Criteria

There are several things to consider when choosing primary and secondary indexes in a parallel processing environment. Because some indexes are chosen based on usage of the data in reporting, there might be some testing needed, later in the development process to arrive at the best possible set of indexes. The primary index of the table does not necessarily need to be a unique index.

Access Demographics: Columns that would appear with a value in a WHERE clause. Choose the column most frequently used for access to maximize the number of direct, single-row access operations. Distribution Demographics: The more unique the index, the better the distribution.

```
PRIMARY INDEX xfact_pr02(pr_nbr)
UNIQUE INDEX xfact_pr01 (src_sys_nm, pr_nbr, pr_line_nbr, acct_asgn_nbr);
```

Figure 4: Primary Index defined on a column most often used as a filter.

Volatility: The data values should not change quite often. Any changes to PI values may result in heavy I/O overhead. Join activity should dictate the PI definition. For large tables, the number of Distinct Primary Index values should be much greater than the number of units of parallelism.

3.5 Synchronizing Source and Target Primary Indexes

Common indexes between source and target tables help bulk inserts. The optimizer performs index-based MERGE JOINs. In a large join operation, a merge join requires less I/O CPU time than nested join. A merge join usually reads each block of the INNER table only once, unless a large number of hash collisions occur. In a real world scenario we noticed that due to missing common primary indexes, the SQL of a stored procedure became 90% skewed. It pulled records from two large tables with several join columns. Run time was 5 hours and 6 minutes to load 9 million rows. After PI synchronization the run-time dropped to 1 minute 11 seconds.

3.6 Deriving Common PI's Between Source Tables

Creating and populating a Global Temporary table helps in avoiding uneven PIs and eliminate LEFT OUTER JOIN in the Final INSERT-SELECT step (second INSERT in Figure 5).

```
INSERT INTO TABLE2.GT_Asset_Cost_Centr
  SELECT AstDrv.Asset_Nbr,AstDrv.Asset_Sub_Nbr
  ,AstDrv.Co_Cd,CC.Prft_Centr_Sap_Nbr
  ,CC.Prft_Centr_Char_Nbr
  FROM Asset.v_asset_DRV  AstDRV
  LEFT OUTER JOIN
  Capital_Analysis.v_dim_cost_centr_curr  CC
  ON AstDRV.cost_centr_cd = CC.cost_centr_sap_nbr;
INSERT INTO
  FROM Asset.v_asset_DRV  AstDRV
  INNER JOIN
  TABLE1.GT_Asset_Cost_Centr CC
  ON AstDrv.Asset_Nbr = CC.Asset_Nbr
AND AstDrv.Asset_Sub_Nbr = CC.Asset_Sub_Nbr
AND AstDrv.Co_Cd = CC.Co_Cd;
```

Figure 5: Deriving a Common PI for Parallel Efficiency.

In a simulation of SQL-run we found that total CPU consumption dropped to 122 second, yielding a 44.68 sec savings. The total I/O operation dropped to 111,192, yielding a 70,632 savings.

3.7 Temporary Tables versus Derived Tables

The solution to some of the resource intensive queries includes conversion of a derived table (DT) to a global temporary table (GTT). This is because the GTT can have statistics collected whereas the DT cannot. The GTT approach makes the optimizer plans more aggressive and rely more heavily on collected statistics as opposed to sampled statistics. As in all of life, there is trade-offs: relying on collected stats would produce better running queries than the random samples. With data skew, the random samples were often wrong and caused wrong choices to be made. We can achieve better performance plans for tables (GTT) with collected statistics. We cannot collect statistics on derived nor volatile tables so these do not perform as well. Statistics collection on join and filter columns improve SQL query performance [6]. Figure 6 shows performance results of an SQL that used derived tables. The result shows that per evaluation criteria the SQL failed in terms of computing resources such as CPU, IO and spool space usage. Their parallel efficiencies are very poor.

SQL Scorecard Results

Project Name: Procure Analysis MSTR
Developer Name: Wally Heaven
System used for Testing: TDCNS1

ETL		CPU Evaluation		I/O Evaluation		Spool Evaluation		EXPLAIN Evaluation		System Rating			
SQL-ID	Procedure Name (SQL)	Total CPU	CPU Parallel Efficiency (%)	Total I/O	I/O Parallel Efficiency (%)	Total Peak Spool	Spool Parallel Efficiency (%)	Statistics on all Joins & Filters?	Joins or Filters on Derived Attributes?	CPU : I/O Ratio	Resource Usage Rating	Parallel Efficiency Rating	Overall Score
1	pr_Fpo_lmt_rpt	1,642	71.65	1,231,559	31.5	5,618,799,616	5.71	YES	NO	1.33	FAIL	FAIL	FAIL

Figure 6:Resource Usage with an SQL that uses derived tables.

Figure 7 shows that each SQL passed in terms of performance evaluation criterion. Computing resources consumption such CPU, IO and spool usage is much lower compared to the resources used shown in Figure 6. Each SQL alsoshows that they higher parallel efficiency.

SQL Scorecard Results

Project Name: Procurment Analysis MSTR
Developer Name: Wally Heaven
System used for Testing: TDCNS1

ETL		CPU Evaluation		I/O Evaluation		Spool Evaluation		EXPLAIN Evaluation		System Rating			
SQL-ID	Procedure Name (SQL)	Total CPU	CPU Parallel Efficiency (%)	Total I/O	I/O Parallel Efficiency (%)	Total Peak Spool	Spool Parallel Efficiency (%)	Statistics on all Joins & Filters?	Joins or Filters on Derived Attributes?	CPU : I/O Ratio	Resource Usage Rating	Parallel Efficiency Rating	Overall Score
1	gt_po_lmt_rpt - TEMP INSERT	7	69.79	53,001	96.18	40,266,240	93.86	YES	NO	0.14	PASS	PASS	PASS
2	gt_po_lmt_rpt1 - TEMP INSERT	14	57.74	59,305	84.05	57,574,240	96.65	YES	NO	0.23	PASS	PASS	PASS
3	gt_po_lmt_rpt2	164	87.53	183,323	85.59	1,634,344,448	60.94	YES	NO	0.89	PASS	PASS	PASS
4	gt_po_lmt_rpt2 - UPDATE	6	64.82	25,538	80.36	350,208	89.06	YES	NO	0.25	PASS	PASS	PASS
5	po_lmt_rpt - FINAL INSERT	9	62.37	12,428	68.5	71,610,368	72.77	YES	NO	0.75	PASS	PASS	PASS

Figure 7: Resource Usage with SQL's that use GTT.

3.8 Handling NULLs for Better Parallel Efficiency

When performing a LEFT OUTER JOIN operation with a column with so many common values performance of join operation degrades. It is important that NULL or blank values be filtered out in the SQL while doing join operation. An MPP (Massively Parallel Processing) machine can get very slow when there is nothing to "parallel process".

```
FROM Procurement.v_funds_rsrv_line AS LINE
LEFT OUTER JOIN
Material.v_mtrl_CRS AS mtrl
ON line.mtrl_id = mtrl.mtrl_id

Results:
-------------
mtrl_id                     Count(*)
000000000000000001            2
NULL/blank                    463950
```

Figure 8: Avoiding NULLs for Parallel Efficiency.

Figure 8 shows a scenario in which case only 2 rows with useful values. The rest of the rows show NULL/BLANK which severely impact database optimizer to perform an MMP.

3.9 Avoiding Updates between Large Tables

When tables are large SELECT/INSERT performs much better. Update is good when source table has fewer rows. An example provided in Figure 9.

```
UPDATE TrgtTbl
  FROM Table1.v_fact_depr_hist_base  TrgtTbl
       ,Table2.gt_fact_depr_hist_anlp4  SrcTbl
  SET calc_curr_qtr_mo1_ordnry_amt = SrcTbl.depr_to_be_post_ordnry_amt
     ,calc_curr_qtr_mo1_unpln_amt = SrcTbl.depr_to_be_post_unpln_amt
  WHERE TrgtTbl.co_cd  = SrcTbl.co_cd AND TrgtTbl.asset_nbr = SrcTbl.asset_nbr
  AND TrgtTbl.asset_sub_nbr = SrcTbl.asset_sub_nbr
AND TrgtTbl.depr_area_cd = SrcTbl.depr_area_cd
  AND TrgtTbl.fscl_yr_mo_nbr_drv1 = SrcTbl.fscl_yr_qtr_mo1_nbr ;
```

Figure 9: Performance Degrades with large volume of Updates.

In one scenario the UPDATE operation by joining large source table caused CPU consumption of 1,013 seconds. If we need to use a subset of data from large tables using global temporary tables will help in computing resource consumption. In a simulation we noticed that by using global temporary tables for a sub-set of data in source table the UPDATE operation took only 600 CPU seconds.

3.10 Partitioned Primary Index

If they are available, Partitioned Primary Indexes (PPI) can be very productive. A PPI is equivalent to row level partitioning. Queries which specify a restrictive condition on the partitioning column avoid full table scans. Larger tables are good candidates for partitioning. The greatest potential gain derived from partitioning a table is the ability to read a small subset of the table instead of the entire table.

Current commercial databases have come up with efficient indexes to improve query performance. When a query is run with filters on PPI columns the DBMS will directly pull data based on particular bucket(s) instead of scanning the whole table. Based on a SQL score-card on both PPI and non-PPI tables it was found that the SQL uses only 33% of the resources to pull rows from a PPI table in relation to a non-PPI table. The run time is also less in the same proportion. The potential gain derived from partitioning a table is the ability to read a small subset of the table instead of the entire table. Queries which specify a restrictive condition on the partitioning column avoid full table scans. By defining a PPI on 'row effective timestamp' the report query performance was found to be four times faster and CPU savings about 33%.

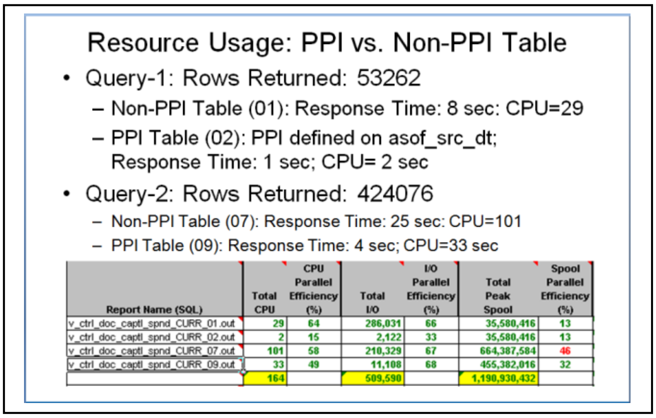

Figure 10: Resource Usage: PPI vs. No PPI tables.

Figure 10 shows a comparison of query response time and computational resource savings between PPI and No-PPI queries. The first query was run to pull 53K rows, with no PPI defined. The response time was eight seconds and CPU consumption was 29 seconds in row one. The same query was run against the same table with PPI defined on row effective date. For the second run the response time was one second and resource consumption was two seconds per row two. The first two rows show the resource usage statistics. A second query was run to pull 424K rows, with no PPI defined. The response time was 25 seconds and resource consumption was 101 CPU seconds in row three. The same query was run against the same table with PPI defined on row effective date. This second run response time was four seconds and resource consumption was 33 seconds in row four.

There are many techniques to improve performance of data warehouse queries, ranging from commercial database indexes and query optimization. A number of indexing strategies have been proposed for data warehouses in literature and are heavily used in practice.

4 SQL Parallel Efficiency and DBMS Resource Usage

In a data warehouse where thousands of queries run by batch processes, analytical and ad-hoc queries and applications all run concurrently, the computing resources are the most precious resources. These computing resources need to be used very efficiently [33] to keep the data warehousing environment stable and running. The analytical community cannot tolerate long running queries or delayed results. Response time of queries is one of the most important indicators of data warehouse stability and its success. The knowledge workers lose confidence in the system if the enterprise data warehouse cannot return information within a reasonable time, especially when it comes to tactical decision making. Transaction latency expressed as a deadline is the most commonly used form of SLA [32], reflecting the user's expectation for the transaction to finish within a certain amount of specified time [41].

In order to ensure the data warehouse is stable, scalable, and queries run efficiently many organizations institute a governing body to oversee the operation and running of the data warehouses. They closely monitor the deployment of objects such as views, stored procedures and macros to make sure they perform efficiently in the data warehouse. In most cases all code that lands on data warehouses goes through a code review process to make sure they are optimized. As a cross-check the DBA (database administrators) team constantly monitors queries and load procedures to make sure the data warehouse is stable and running efficiently. Some things to watch for, to help with parallel efficiency:

4.1 Large Distribution Steps

Occasionally, there will be a job with a step which takes a lot of resources, just to get two tables ready for a join step. This happens (as mentioned earlier) when the two tables being joined do not have the same Primary Index (PI). Sometimes, one of the tables can be changed, so the PI's are the same. When the PI's are the same, the large redistribution step is eliminated, sometimes with very nice results. However, we need to be careful not to introduce too much non-uniqueness in the PI. Table 1 shows a real example of such results. The first row shows that almost all of the resources used on this job were spent in the redistribution step. However, once redistribution steps are eliminated most of theresource use disappeared (second row).

Table 1
Resource Saving Avoiding Row Redistribution

	Elapsed Time	I/O	CPU	Spool
before	00:05.1	00:00.0	31:12.0	510,038,016
after	00:00.1	00:00.0	09:36.0	0

4.2 Secondary Indexes

Sometimes we cannot just change the PI of a table for the purposes of helping a join step. There are many reasons for choosing the PI of a table. The first criteria should be Reporting Access Requirements. In these cases, it is possible to add a Secondary Index, the same as what we would have liked for a PI.

Table 2
Computing Resource Saving using Secondary Index

BEFORE					
LogDate	QryCount(*)	AvgElapsedTime	AvgCPUTime	AvgSpoolUsage	AvgIOCount
10/8/2012	717	00:02.2	8.70	503,352,026	88,810
AFTER					
LogDate	QryCount(*)	AvgElapsedTime	AvgCPUTime	AvgSpoolUsage	AvgIOCount
11/12/2012	717	00:01.1	0.27	1,639,843	13,663

In one such case, we had a lot of queries doing the same join, so we added a Secondary Index to a table. This helped the JOIN Condition find the applicable rows faster. This is bit different from an Index which helps find the rows being selected.

Our experiment shows that resource reduction for one day was 6,000 CPU seconds, spool space 350 gigabytes, and I/O reduction 50 million. Elapse time was 15 minutes.

4.3 Partitioning Rows which are Accessed Often

In this next case, we found that a lot of queries were asking for rows within a given date range, so we added Partitioning, in a way that reduced tables scans to a smaller set of data-blocks. Table 3 represents a set of queries for 1 day's activity.

Table 3
Using Partitioning to Avoid or Reduce Table-Scans

Log Date	UserName	Before Query Count	After Query Count	Before Avg CPU	After Avg CPU	CPU Saved
6/20/2011	MMBIETL1	14,037	15,535	0.25	0.23	286
6/20/2011	A341545	7,154	10,724	0.36	0.07	3111
6/20/2011	MMBIAP01	6,339	6,298	9.48	8.09	8771
6/20/2011	A130330	2,224	2,344	1.28	0.96	759
6/20/2011	MMBI001	2,115	2,314	46.93	43.06	8956
6/20/2011	A1887155	9	57	63.03	16.96	2626
						24510

4.4 Skewed Processing

Sometimes we chose a certain PI to help reporting, but allows too many rows to be stored on 1 or just a few AMPs (Units of Parallelism). We discussed Skew in a previous section. In this case, we can use a different PI to spread the data more evenly, then build a Secondary Index where we removed the first Primary Index.Here is a case where we did this. We use a bit more resource with a Secondary Index over a Primary, but we save a lot more than that by spreading the work out over the AMPS more evenly.

In Figure 11, we can see where we installed the change at 14:00 hours: There is a set of jobs which run every hour. We can see the reduction in resources for the next hours.

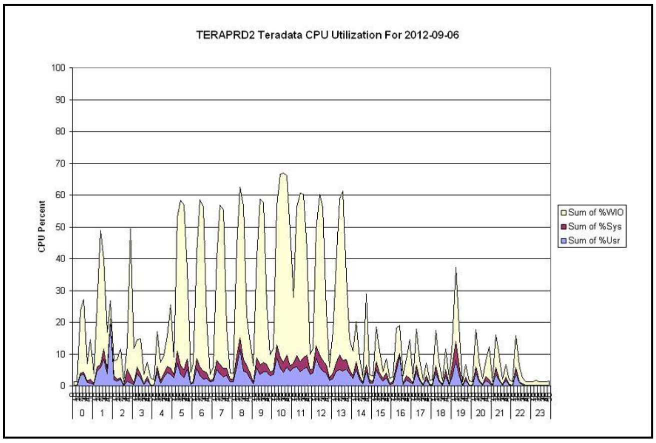

Figure 11: Resource Saving by Improving Parallel Efficiency.

4.5 Using Set versus Multi-Set Tables

SET tables do not allow duplicate rows – Multi-set tables do. The combination of a SET table, and a Non-Unique Primary Index, can dangerous if there is no other uniqueness constraint on the table (such as a Unique Secondary Index). If there is a Unique Secondary Index, the table does not need to "worry" about checking for duplicate rows (because the Index will be checking). If there is no other unique constraint on the table, when we have multiple rows with that same PI, as rows are inserted, the table needs to check all rows with the same PI to see if in fact the whole row is a duplicate. If it is a full row duplicate, it will not be allow to be inserted. This dupe-row-checking can get very expensive if there are a lot of rows with the same (non-unique) PI.

Table 4
Resource Savings – Set vs. Multi-Set tables

BEFORE LogDate	TotalIO	TotalCPU	StartTime	ElapsedTime	ImpactCPU	CpuSkew	SpoolUsage	Table being worked
4/8/2013	183,104.00	13.3	28:39.1	0:00:23.17	254.74	19.16	33,772,544	IMM_FCST_INSTALLATION_MV
4/8/2013	1,079,940.00	53.83	29:19.1	0:01:16.37	2,244.10	41.69	150,929,408	IMM_FCST_REQUEST_MV
4/8/2013	228,405.00	13.88	29:03.7	0:00:14.53	250.7	18.07	32,450,048	IMM_FCST_REMOVAL_MV
4/8/2013	943,791.00	36.86	30:48.0	0:00:55.51	1,520.78	41.26	73,801,728	IMM_FCST_ISSUE_MV
4/8/2013	190,319.00	10.06	30:36.2	0:00:10.95	234.86	23.35	41,681,920	IMM_FCST_RETURN_MV
	2,613,562.00		Totals	0:03:05.21	4505.18	Avg 43.53		
AFTER LogDate	**TotalIO**	**TotalCPU**	**StartTime**	**ElapsedTime**	**ImpactCPU**	**CpuSkew**	**SpoolUsage**	Table being worked
4/22/2013	123,034.00	3.76	27:30.7	0:00:02.57	7.49	1.99	21,797,376	IMM_FCST_RETURN_MV
4/22/2013	119,611.00	5.07	27:20.7	0:00:03.63	28.22	5.57	74,287,616	IMM_FCST_REMOVAL_MV
4/22/2013	347,427.00	8.3	27:25.1	0:00:04.86	30.96	3.73	113,038,848	IMM_FCST_REQUEST_MV
4/22/2013	312,617.00	8.71	27:34.2	0:00:04.99	18.58	2.13	117,903,360	IMM_FCST_ISSUE_MV
4/22/2013	176,712.00	6.66	27:15.3	0:00:04.55	47.66	7.15	89,515,008	IMM_FCST_INSTALLATION_MV
	1,077,901.00		totals	0:00:20.65	132.91	Avg 4.11		
Impove ments	2X			9X	33X	10X		

Let us suppose, there are 1000 rows with the same PI to be inserted. The second row inserted needs to only check 1 row for duplicates. The 100th row needs to check 99 rows. The 950th row needs to check 949 rows. The number of row checks would be 1+2+3+4+5…+999. Here are the results of a set of changes we made to a set of jobs which populated a few tables. We changed them from set to multi-set tables, to avoid duplicate row checking. And we changed the Primary Indexes to a column with less skewing. And we added a Secondary Index to replace the benefit we had with the old Primary Index.

5 Measuring SQL Performance

Performance statistics of SQL blocks in a stored procedure are shown in Figure 10. We show the score-card results of a stored procedure SQL blocks. The SQL's were written in such a way that they are in compliant with the parallel processing architecture of the underlying database system.

In Figure 12 we can see that each of the SQL's in a stored procedure passed in terms of CPU, I/O, and spool parallel efficiency. In the stored procedure SQL's were written in small code blocks. Each of them takes fewer CPU seconds; they run with parallel efficiency. We see all of the SQL blocks passed in score-carding.

SQL Scorecard

Project Name: Capital_DRV
Developer Name: Shameem Akhter
System used for Testing: TDEDW1

ETL		CPU Evaluation		I/O Evaluation		Spool Evaluation		EXPLAIN Evaluation		System Rating			
SQL-ID	Report Name (SQL)	Total CPU	CPU Parallel Efficiency (%)	Total I/O	I/O Parallel Efficiency (%)	Total Peak Spool	Spool Parallel Efficiency (%)	Statistics on all Joins & Filters?	Joins or Filters on Derived Attributes?	CPU : I/O Ratio	Resource Usage Rating	Parallel Efficiency Rating	Overall Score
1	pr_Ffact_grant_grir	4	62.56	22,064	94.97	34,054,656	64.87	YES	NO	0.18	PASS	PASS	PASS
2	pr_Ffact_grant_grir1	55	75.29	245,882	81.13	438,595,072	88.82	YES	NO	0.22	PASS	PASS	PASS
3	pr_Ffact_grant_grir2	3	54.77	22,488	85.49	503,808	85.42	YES	NO	0.11	PASS	PASS	PASS
4	pr_Ffact_grant_grir3	2	43.59	19,391	96.65	16,996,352	93.46	YES	NO	0.11	PASS	PASS	PASS
5	pr_Ffact_grant_grir4	2	57.75	15,148	87.18	4,651,008	81.57	YES	NO	0.13	PASS	PASS	PASS
6	pr_Ffact_grant_grir5	3	55.3	60,780	70.5	986,112	71.65	YES	NO	0.04	PASS	PASS	PASS
7	pr_Ffact_grant_grir6	4	61.53	63,890	71.18	986,112	71.65	YES	NO	0.06	PASS	PASS	PASS
8	pr_Ffact_grant_grir7	4	55.26	25,329	92.58	47,062,528	84.36	YES	NO	0.18	PASS	PASS	PASS
9	pr_Ffact_grant_grir8	2	65.89	15,855	86.02	1,014,784	71.19	YES	NO	0.13	PASS	PASS	PASS
10	pr_Ffact_grant_grir9	2	63.28	14,886	90.15	1,015,296	71.23	YES	NO	0.13	PASS	PASS	PASS
11	pr_Ffact_grant_grir10	3	58.56	27,569	96.05	17,804,288	93.12	YES	NO	0.11	PASS	PASS	PASS
12	pr_Ffact_grant_grir11	3	40.56	24,896	86.73	17,804,288	93.12	YES	NO	0.11	PASS	PASS	PASS
13	pr_Ffact_grant_grir12	3	51.19	66,406	72.66	1,973,760	83.66	YES	NO	0.04	PASS	PASS	PASS
14	pr_Ffact_grant_grir13	2	50.43	40,634	73.61	1,973,760	83.66	YES	NO	0.06	PASS	PASS	PASS
15	pr_Ffact_grant_grir14	2	65.49	15,162	88.73	1,015,296	93.89	YES	NO	0.13	PASS	PASS	PASS
16	pr_Ffact_grant_grir15	2	66.67	15,743	88.17	1,981,440	83.98	YES	NO		PASS	PASS	PASS
17	pr_Ffact_grant_grir16	2	49.53	14,415	84.36	1,981,440	83.98	YES	NO		PASS	PASS	PASS
18	pr_Ffact_grant_grir17	1	40.63	4,981	82.36	11,043,840	87.09	YES	NO		PASS	PASS	PASS
19	pr_Ffact_grant_grir18	2	45.94	8,798	91.65	8,102,912	83.26	YES	NO		PASS	PASS	PASS
20	pr_Ffact_grant_grir19	2	63.67	14,421	81.64	29,583,360	90.92	YES	NO		PASS	PASS	PASS
21	pr_Ffact_grant_grir20	2	36.89	7,740	94.85	19,288,064	94.56	YES	NO	0.31	PASS	PASS	PASS
		105		746,478		658,418,176		...	...	0.14	PASS		

Figure 12: Score-Card Results of SQL's of a Stored Procedure

6 Conclusion

In this article we provided an overview of a parallel processing DBMS architecture. We have highlighted as to what key aspects needs to be considered to take advantage of parallelism. We have provided an exhaustive list of techniques of SQL optimization. These techniques have been tested and implemented in a large production data warehouse system. In each of the optimization techniques we have provided computing resource savings as well query response time decrease statistics.

We proposed evaluating SQL queries using SQL scorecard tools. A scorecard process and performance optimization techniques will enable the SQL programmers to empower themselves in writing efficient queries without much dependence on database administrators. Currently, database administrators spend many hours inspecting various log files and queries [48]. Our proposed developer-centric SQL query optimization will help database administrators maintain a stable database system and its performance with much less effort. As part of our future research we intend to do explore optimization of queries in NoSQL database systems.

Acknowledgment

An earlier version of this article was presented at the IEEE 26th Canadian Conference of Electrical and Computer Engineering (CCECE 2013), Regina, Saskatchewan, Canada, May 5 - 8, 2013 [14].This article has been substantially revised and enhanced, based on comments received during conference presentation and comments received from anonymous reviewers of this journal. The author would like to thank the anonymous reviewers for their constructive comments provided on earlier drafts of this article.

References

[1] N. Rahman, "SQL Scorecard for Improved Stability and Performance of Data Warehouses," International Journal of Software Innovation (IJSI), Vol. 4, No. 3, pp. 22-37, July- September 2016.

[2] S. Akhter and N. Rahman, "Building a Customer Inquiry Database System," International Journal of Technology Diffusion (IJTD), Vol. 6, No. 2, pp. 59–76, April-June 2015.

[3] G. Allen and J. Parsons, "Is Query Reuse Potentially Harmful? Anchoring and Adjustment in Adapting Existing Database Queries," Information Systems Research, Vol. 21, No. 1, pp. 56-77, 2010.

[4] R. Armstrong, "When and Why to Put What Data Where," Teradata Corporation White Paper, 1-5, 2007.

[5] N. Rahman, "Saving DBMS Resources While Running Batch Cycles in Data Warehouses," International Journal of Technology Diffusion (IJTD), Vol. 1, No. 2, pp. 42–55, April-June, 2010.

[6] C.G. Corlatan, MM. Lazar, V. Luca and O.T. Petricica, “Query Optimization Techniques in Microsoft SQL Server,” Database Systems Journal, Vol. 5, No. 2, pp. 33-48., 2014.

[7] U. Dayal, H. Kuno, J.L. Wiener, K. Wilkinson, A. Ganapathi and S. Krompass, "Managing Operational Business Intelligence Workloads," ACM SIGOPS Operating Systems Review archive, Vol. 43, No. 1, pp. 92-98, 2009.

[8] S. Deepak, S.U. Kumar, M Durgesh and K.P. Bhupendra, "Query Processing and Optimization of Parallel Database System in Multi-processor Environments," In proceedings of the 2012 Sixth Asia Modelling Symposium (pp. 191-194).

[9] N. Rahman, "Measuring Performance for Data Warehouses - A Balanced Scorecard Approach," International Journal of Computer and Information Technology (IJCIT), Vol. 4, No. 1, pp. 1–7, January-March, 2013.

[10] E.W. Dempster, N.T. Tomov, M.H. Williams, H. Taylor, A Burger, P. Trinder and P. Broughton, "Modelling Parallel Oracle for Performance Prediction," Distributed and Parallel Databases, Vo. 13, No. 3, pp. 251–269, 2003.

[11] D. Dey, Z. Zhang and P. De, "Optimal Synchronization Policies for Data Warehouse," Information Journal on Computing, Vol. 18, No. 2, pp. 229-242, 2006.

[12] S. Elnaffar, P. Martin, B. Schiefer and S. Lightstone, "Is It DSS or OLTP: Automatically Identifying DBMS Workloads," Journal of Intelligent Information Systems, Vol. 30, No. 3, pp. 249–271, 2008.

[13] J. Evermann, "An Exploratory Study of Database Integration Processes," IEEE Transactions on Knowledge and Data Engineering, Vol. 20, No. 1, 2008.

[14] N. Rahman, "SQL Optimization in a Parallel Processing Database System," In proceedings of the IEEE 26th Canadian Conference of Electrical and Computer Engineering (CCECE 2013), Regina, Saskatchewan, Canada, May 5 - 8, 2013.

[15] A. Ferrández, A. Maté, J. Peral, J. Trujillo, E.D. Gregorio and M.-A. Aufaure, "A Framework for Enriching Data Warehouse Analysis with Question Answering systems," Journal of Intelligent Information Systems, Vol. 46, No. 1, pp. 61-82, 2016.

[16] S. Ganguly, W. Hasan and R. Krishnamurthy, "Query Optimization for Parallel Execution," In Proceedings of the 1992 ACM SIGMOD international conference on Management of data, San Diego, California, United States, pp. 9 – 18, 1992.

[17] García-García and C. Ordonez, "Extended Aggregations for Databases with Referential Integrity Issues," Data & Knowledge Engineering, Vol. 69, pp. 73–95, 2010.

[18] N. Rahman, "Temporal Data Update Methodologies for Data Warehousing," Journal of the Southern Association for Information Systems (JSAIS), Vol. 2, No. 1, pp. 25–41, 2014.

[19] T. Georgieva, "Discovering Branching and Fractional Dependencies in Databases," Data & Knowledge Engineering, Vol. 66, pp. 311-325, 2008.

[20] A. Ghazal, D. Seid, R. Bhashyam, A. Crolotte, M. Koppuravuri and G, Vinod, "Dynamic Plan Generation for Parameterized Queries," In Proceedings of SIGMOD’09, Providence, RI, USA.

[21] M. Golfarelli and S. Rizzi, "Data Warehouse Design: Modern Principles and Methodologies," McGraw-Hill Osborne Media; 1 edition, May 26, 2009.

[22] G. Hill and A. Ross, "Reducing Outer Joins," The VLDB Journal, Vol. 18, pp. 599–610, 2009.

[23] A. Karakasidis, P. Vassiliadis and E. Pitoura, "ETL Queues for Active Data Warehousing," In Proceedings of the 2nd International Workshop on Information Quality in Information Systems, IQIS 2005, Baltimore, MD, USA.

[24] N. Rahman and D. Rutz, "Building Data Warehouses Using Automation," International Journal of Intelligent Information Technologies (IJIIT), Vol. 11, No. 2, pp. 1–22, April-June, 2015.

[25] M.A. Kashem, A.S. Chowdhury, R. Deb and M. Jahan, "Query Optimization on Relational Databases for Supporting Top-k Query Processing Techniques," International Journal of Computer and Information Technology (IJCIT), Vol. 1, No. 1, pp. 53-58, 2010.

[26] A. Kerkad, L. Bellatreche, P. Richard, C. Ordonez and D. Geniet, "A Query Beehive Algorithm for Data Warehouse Buffer Management and Query Scheduling," International Journal of Data Warehousing and Mining, Vol. 10, No. 3, pp. 34–58, 2014.

[27] R. Kimball and M. Ross, "The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling," 3rd edition, Wiley. 2013, Hoboken, New Jersey, USA.

[28] S. Krompass, H. Kuno, U. Dayal and A. Kemper, "Dynamic Workload Management for Very Large Data Warehouses – Juggling Feathers and Bowling Balls," In Proceedings of the 33rd international conference on Very large data bases, Vienna, Austria, 1105-1115, 2007.

[29] S. Krompass, H. Kuno, J.L. Wiener, K. Wilkinson, U. Dayal and A. Kemper, "Managing Long-Running Queries," In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, Saint Petersburg, Russia, pp. 132-143, 2009.

[30] D. Lee, S.K. Cha and A.H. Lee, "A Performance Anomaly Detection and Analysis Framework for DBMS Development," IEEE Transactions on Knowledge and Data Engineering, Vol. 24, No. 8, pp. 1345-1360, 2012.

[31] N. Rahman, D. Rutz, S. Akhter and F. Aldhaban, "Emerging Technologies in Business Intelligence and Advanced Analytics," ULAB Journal of Science and Engineering (JSE), Vol. 5, No. 1, pp. 7–17, November 2014.

[32] P. Leitner, J. Ferner, W. Hummer and S. Dustdar, "Data-driven and Automated Prediction of Service Level Agreement Violations in Service Compositions," Distrib Parallel Databases, Vol. 31, pp. 447-470, 2013.

[33] M. Mannino, S.N. Hong and I.J. Choi, "Efficiency Evaluation of Data Warehouse Operations," Decision Support Systems, Vol. 44, pp. 883-898, 2008.

[34] Y. Meng, P. Bird, P. Martin and W. Powley, "An Approach to Managing the Execution of Large SQL Queries," In proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research, Richmond Hill, Ontario, Canada (pp. 268-271).

[35] V. Narasayya, I. Menache, M. Singh, F. Li, M. Syamala and S. Chaudhuri, "Sharing Buffer Pool Memory in Multi-tenant Relational Data-

base-as-a-service," In proceedings of the VLDB Endowment, Vol. 8, No. 7, pp. 726–737, 2015.
[36] N. Rahman, "An Empirical Study of Data Warehouse Implementation Effectiveness," International Journal of Management Science and Engineering Management (IJMSEM), February 2016.
[37] T. Neumann, "Engineering High-Performance Database Engines," In proceedings of the VLDB Endowment, Vol. 7, No. 13, pp. 1734–1741, 2014.
[38] C. Ordonez and Z. Chen, "Horizontal Aggregations in SQL to Prepare Data Sets for Data Mining Analysis," IEEE Transactions on Knowledge and Data Engineering, Vol. 24, No. 4, pp. 678–691, 2012.
[39] R. Osman and W.J. Knottenbelt, "Database System Performance Evaluation Models: A survey, Performance Evaluation," Vol. 69, No. 10, pp. 471-493, 2012.
[40] W. Powley, P. Martin and P. Bird, "DBMS Workload Control Using Throttling: Experimental Insights," In proceedings of the 2008 conference of the center for advanced studies on collaborative research, Ontario, Canada.
[41] M.A. Sharaf and P.K. Chrysanthis, "Optimizing I/O-Intensive Transactions in Highly Interactive Applications," In Proceedings of the 35th SIGMOD international conference on Management of data, Providence, Rhode Island, USA, 785-798, 2009.
[42] F.D. Tria, E. Lefons, and F. Tangorra, "Benchmark for Approximate Query Answering Systems," Journal of Database Management, Vol. 26, No. 1, pp. 1–29, 2015.
[43] N. Rahman, N. Kumar and D. Rutz, "Managing Application Compatibility During ETL Tools and Environment Upgrades," Journal of Decision Systems (JDS), Vol. 25, No. 2, pp. 136-150. April-June 2016.
[44] D. VanderMeer, K. Dutta and A. Datta, "A Cost-based Database Request Distribution Technique for Online E-commerce Applications," MIS Quarterly, Vol. 36, No. 2, pp. 479-507, 2012.
[45] W. Weill, M. Subramani and M. Broadbent, "Building IT Infrastructure for Strategic Agility," MIT Sloan Management Review, 2002.
[46] B. Wixom and H. Watson, "The BI-Based Organization," International Journal of Business Intelligence Research, Vol. 1, No. 1, pp. 13-28, 2010.
[47] S. Wu, F. Li, S. Mehrotra and B.C. Ooi, "Query Optimization for Massively Parallel Data Processing," In proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11), Cascais, Portugal, 2011.
[48] D.Y. Yoon, B. Mozafari and D. Brown, "DBSeer: Pain-Free Database Administration through Workload Intelligence," In proceedings of the VLDB Endowment, Vol. 8, No. 12, pp. 2036–2039, 2015.
[49] N. Rahman, "Enterprise Data Warehouse Governance Best Practices," International Journal of Knowledge-Based Organizations (IJKBO), Vol. 6, No. 2, pp. 21-37. April-June 2016.

Nayem Rahmanis a Senior Enterprise Application Developer in Cross Enterprise Systems IT, Intel Corporation. He has implemented several large projects using data warehousing and big data technologies for Intel's mission critical enterprise DSS platforms and solutions. He is currently working toward the Ph.D. degree in the Department of Engineering and Technology Management, Portland State University, USA.He holds an M.S. in Systems Science (Modeling & Simulation) from Portland State University, Oregon, USA and an MBA in Management Information Systems (MIS), Project Management, and Marketing from Wright State University, Ohio, USA. He has published more than 30 articles in peer-reviewed international jounrals and conference proceedings. He has four book chapters published. He has presented his creative work at many industry and academic conferences in USA and Canada.His principal research interests include Big Data Analytics, Big Data Technology Adoption, Data Mining for Business Intelligence, and Simulation-based Decision Support System (DSS).

Leonard Sutton is an Independent SQL Performance Tuning Consultant, with a focus on Massively Parallel Processing (MPP) systems. He is currently working at Boeing as an Independent Consultant.He is a certified Teradata Master. His work experience include 14 years at Nike Inc. (DB2 and Teradata), 11 years at Teradata Corporation as a Consultant helping many different Customers and 2 years with the Teradata Performance Tuning COE team. He has solid experience and knowledge about what DOES and DOES NOT work on Large Data Warehouse Systems using Parallel Processing Architecture.