International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 10 (2018) pp. 038–046

Database Tuning Using Oracle Materialized View for Manufacturing Industry

Norazah Md Khushairi[1], Nurul A. Emran[2] and Anil Kumar Menon[3]

[1]Silterra Malaysia Sdn Bhd,Kedah, Malaysia
norazah_khushairi@silterra.com

[2]Universiti Teknikal Malaysia, Melaka,Malaysia
nurulakmar@utem.edu.my

[3]Oracle, Oracle Corporation, Malaysia
govindan.ak.menon@oracle.com

Abstract: **The need to maintain database performance in Silterra Malaysia is crucial as data produced from complex manufacturing processes must be recorded in timely manner for reporting purposes. Query rewriting using Oracle Materialized View (MV) is one form of corrective action adopted by Silterra in order to tune its database, which is usually affected by problematic SQLs. However, whether MVs are useful in most cases of query rewriting is an open problem. In this paper, the flow of SQL query rewriting process using MVs is presented. Steps to identify problematic SQLs and to rewrite them are given based on DBA's experience in dealing with database performance issue in this industry. The result of using MVs shown using real fabrication data in Silterra reveals that, even though most MVs perform better than queries without MVs, there are cases that require alternative for MVs.**
Keywords: query rewriting, materialized view, database tuning

I. Introduction

The challenge of handling large volume of real-time and persistent data in databases has become the attention in data management community for decades [1]. In most industries, large volumes of raw data are collected, accumulated and transformed into huge datasets and prepared (through data cleaning stages) before it can be further used for reporting and analytics. Large data volume can cause overhead to Database Management System (DBMS) that usually causes database performance issues [2]. The performance overhead is faced by many manufacturing industries especially those that rely on complex processes in producing their commercial products.

For example, manufacturing industry for semiconductors that relies on the most complex manufacturing process in the world (refer [3],[4]) is among the industries that is affected by poor database performance. Among the semiconductor processes, wafer fabrication is the most complex process that requires the highest capital investment [32]. It takes about 50 to 70 days to process 40,000 to 50,000 Work-In-Process (WIP) involving 400 equipments and between 300 to 900 steps to complete the cycle [5], [7], [8]. Furthermore, there is a demand to fabricate a product within a tight time frame since competitive cycle time is the main factor for business success [5]. Indeed, reduction of this cyclic time frames will shorten product time to market, boost up throughput, decrease operational costs and strengthen customer's confidence [5]. Therefore, in this industry, it is crucial to observe the wafer production cycle time closely. Each activity, movement, transaction, and processing history needs to be closely monitored for quality assurance and for shortening development cycle time. Furthermore, semiconductor processes are delicate and require close monitoring that is beyond human's capabilities [6].

Semiconductor industry evolves in a complex environment in which strategic IT alignment must be ensured [9]. Thus, effective reporting tools need to be adopted for monitoring. However, poor database performance can affect the effectiveness of reporting tool. Database performance is a crucial issue as it becomes the barrier for DBMS to promptly response to the reporting tool's query. This poor performance of the monitoring tools affect production's cycle time and therefore the industry's productivity and profits generation. The top management in this industry relies on immediate reporting result in order to effectively manage the competitive fabrication cycle time. Thus, database performance issue is crucial for this industry. One of the main causes of database performance issue is slow query due to problematic Structure Query Language (SQL).

In this paper we will explore the problem of reporting delay caused by SQLs in Silterra Malaysia Sdn Bhd, (a Wafer Semiconductor Factory). In particular, we will reveal how Silterra deals with this problem. We present related works on database workload issue related to SQL query and how to identify the problematic SQL query in Section 2. Focus is given to tune the problematic SQL using SQL rewriting process in Section 3. In Section 4 we present how SQL query performance is improved using MV. Section 5 presents the improvement analysis by using MVs. Section 6 concludes the paper.

II. Identifying Problematic SQL Query

In monitoring database performance, main indicators to look for are: executed reports that exceed the average run times and data loads that consume more resources (or time) than average. There are many possible causes of poor database performance such as large tables, poorly implemented database design or badly rewritten code. The common initial step taken is to identify the possible areas of performance problem. These areas are operating system, hardware, database or application. Performance tuning can be performed at hardware level, instant workload, instant object or SQL statement [10]. In this paper, we will explore the performance tuning at application level since application level tuning can bring better performance results as compared to other levels [11]. In this level, we need to find inefficient or high load run-time SQL statements. These run-time SQL statements, might consume high database time, longer time for completion and therefore causing performance degradation.

Identifying problematic SQL query in database environment is a fundamental step of SQL query tuning. It involves identifying high load or top low performant SQL statements, improving the SQL execution plan produced by optimizer and finally, implementing the corrective actions [12]. The easiest way to accomplish this is by periodically collecting SQL execution statistics and analyzing the results [13].

With Oracle Database, the Database Administrator (DBA) can use the Oracle Enterprise Manager (OEM) which provides functions such as top activity information (along charts), top sessions, and top SQL queries. This product will list down the run-time queries that consume longer completion time and will assist the DBA to quickly identify the SQL statements that are responsible for performance degradation at a specific point of time. OEM also provides a tool to identify SQL queries that consume high DB time, long running SQLs and operations, and SQLs with execution plan changes. Automatic Database Diagnostic Monitor (ADDM) is a tool commonly used to identify the top SQLs and to study the impact. It also will show the SQL's occurrence frequency and will identify the possible root causes. The example of root causes are poorly written SQL with bad execution plan (i.e., full table scan, cartesian join and incorrect index).

DBAs also can create a custom script to expedite the search of inefficient SQL statements that need to go through the tuning process. Should it identify that poor performance is caused by inefficient SQL queries, these queries are used as samples in tuning process.

III. SQL Rewriting Process

Based on literature and observation, we present a methodology for current practice adopted in SQL tuning process as shown in Figure 1. This diagram is used as a guideline for the tuning process that is normally performed whenever database performance problem is detected.

SQL tuning starts with DBAs collecting monitoring data from the production database in an attempt to diagnose the problems. At this stage, any SQL statement that needs to undergo the SQL tuning flow will be identified.

In the second step, the execution plan that describes the detailed steps necessary to execute the SQL statement is analyzed to diagnose the cause of the problem. The `Explain Plan` command is used to display the execution plan report. SQL statement performance depends heavily on the optimal execution of plans generated by the query optimizer with the unenviable task of generating efficient SQL statement [14].

By analyzing the plans, DBAs are able to get more detailed information to understand which plan is selected by optimizer and the reason of the selection. From this plan, expert DBAs propose solutions for the problematic queries and decide whether the queries need to be rewritten or not.

The success of SQL tuning usually depends on skilled DBAs or the laborious trial-and-error steps [15]. DBAs and developers will try to tune SQL statement based on the execution plan or other solutions. As shown in Figure 1, coding process consists of Rewrite, Execute, Verify and Apply. The coding process will be repeated until a solution is found.

The third step, which is the rewrite step will be initiated once the decision to rewrite is made by the DBAs. There are several ways to rewrite SQL queries. One way to rewrite is by fixing wrongly written SQL committed by application developers. Normally, we will try to tune or rewrite the SQL using the common case. For example a developer does not use the indexed field correctly or no index is used at all. If a table has high cardinality, index is useful to avoid full table scan. Using fields that are indexed in the WHERE clause of the SQL statement will improve the performance faster than using non-indexed fields. However, in some cases, simple solution like this cannot solve the problem. In this case, further investigation is required before the query can be rewritten.

After the query execution, verification step is made to check the accuracy of query results and performance's improvement. If the result of the query is correct, and the performance has shown acceptable improvement, the new corrective version of SQL query will be applied. Otherwise, the query needs to be rewritten. The coding process stage will be repeated until acceptable results are yielded.

In Silterra, several methods have been adopted to tune problematic SQL queries as shown in Figure 2. As database performance degrades with increasing data volumes, the common practices are to ensure proper use of index for table fields, table partitioning, and query rewriting using materialized views (MVs). It is important for a developer to know how to write an efficient SQL because inefficient SQLs in production can highly impact the overall performance of the database and the downstream applications. Most of the methods are performed by DBAs, assisted by application developers. DBAs need to perform regular inspection and database performance diagnosis to identify the top ten activities that cause the problems before corrective method can be applied. DBAs need to decide which method is suitable based on the problems. Each of the method has its own advantages and limitations in database performance tuning. One of the most successful performance tuning steps is to rewrite the SQL using MVs. Among the adopted methods, query rewriting using MVs has recorded more promising performance.

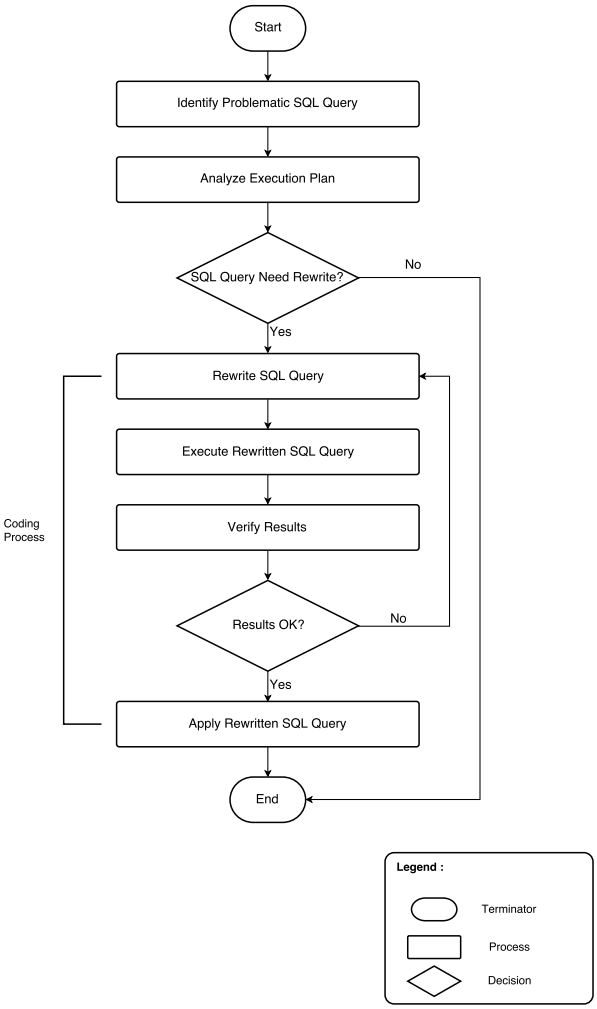

Figure. 1: Common SQL Query Rewriting Flow

Similar flow as shown in Figure 1 is adopted in Silterra to tune slow SQL queries. MV is a well-known optimization strategy with potential improvements in query processing time [17]. Implementation of MVs in Silterra's Manufacturing Executing System (MES) reporting function so far has successfully reduced the time taken for queries completion especially for queries against large tables. In the next section, we present the MV concept and it's role in dealing with slow SQL queries.

IV. Materialized View

MVs are physical database objects[18]. It is a database object that uses a query script and executes it at a specific time, and store the results of the query in the storage [19]. Even though the name MV has a "view" in it, it is not a virtual table like views. The similarities are in terms of keeping the structure of the SQL query being embedded in the objects. However views only keep the structure but MVs keep the both, structure and the data. MVs are similar to normal views, but the result of query has been stored into table [18]. Views are only executed when used as part of a SELECT statement whereas MVs will persist the results of query in the storage. The MV's query is executed and refreshed during

Table 1: Selected Research in Materialized Views

Researcher Findings	Authors
Modified strategy of group query based on MV (GQMV) by making full use of the star schema feature from the aspect of improving searching capability.	Li Guodong, Wang Shuai, and Liu Chang'an. 2010 [18]
Materialized reporting function views which rewrite queries with reporting functions as well as aggregation queries.	Dirk Habich, Wolfgang Lehner, and Michael Just Dresden, 2006 [19]
XPath Queries rewriting using MVs with an algorithm for finding minimal rewritings.	Wanhong Xu, and Z. Meral Ozsoyoglu, 2005 [20]
Answering queries using MVs with minimum size can reduce the size of the relations needed to compute the query answer.	Chirkova, Rada Li, Chen Li and Jia, 2006 [21]

specified time and the data will be kept physically until next refresh. MV is like a cache, a copy of data that can be promptly accessed [23]. Oracle defines MV as a replica of a master target from a single point in time where replication tables are continuously updated by other master tables [27]. MV's data will be updated by refreshing them after changes made to the base tables, either by using incremental or complete refresh method. After its creation, MV can be used to query the data same as the base tables. Some contribution of MVs are as shown in Table 1.

Figure 3 shows the logical diagram of view materialization process as proposed by Karde and Thankare (2010) [24]. There are three layers involved, which are the base table, process to materialize and the MV. Base table become a source data for MV and the view selector will process it based on the configuration set up during MV creation. The architecture of view materialization process shows the view selector interacts with the query processor (QP), that will be refreshed based on the query updates. The relevant set of data will be selected based on the given set of queries. The layer between view and MV shown in the diagram is a process to materialize the source of the query. Moreover, there is a methodological layer to determine what kind of views will be beneficial under various situations such as selectivity, complexity and database size including the maintenance cost consideration [20]. The common MV management activities are identifying the MVs to be created, ensuring that MVs properly refreshed, determining the MVs effectiveness, measuring the MV space, and preforming MV housekeeping (such as dropping all the unused MVs) [25][23].

Based on the problematic SQL list, the DBAs or developers will decide which table needs to be materialized by setting the timing for queries updates. The refresh time setting is specified during the MV creation. Whenever the queries update time is triggered, view selector will process the queries and perform data retrieval selection. The selection on which source of data to be materialized will be systematically made to achieve the purpose of getting the best query performance from a given SQL environment workload. Karde and Thakare (2010) stated that selecting views to materialize for the purpose of efficiently supporting the decision making is one of the important decisions in designing data warehouse [20].

In Silterra, the need for fast data retrieval is driven by rapid

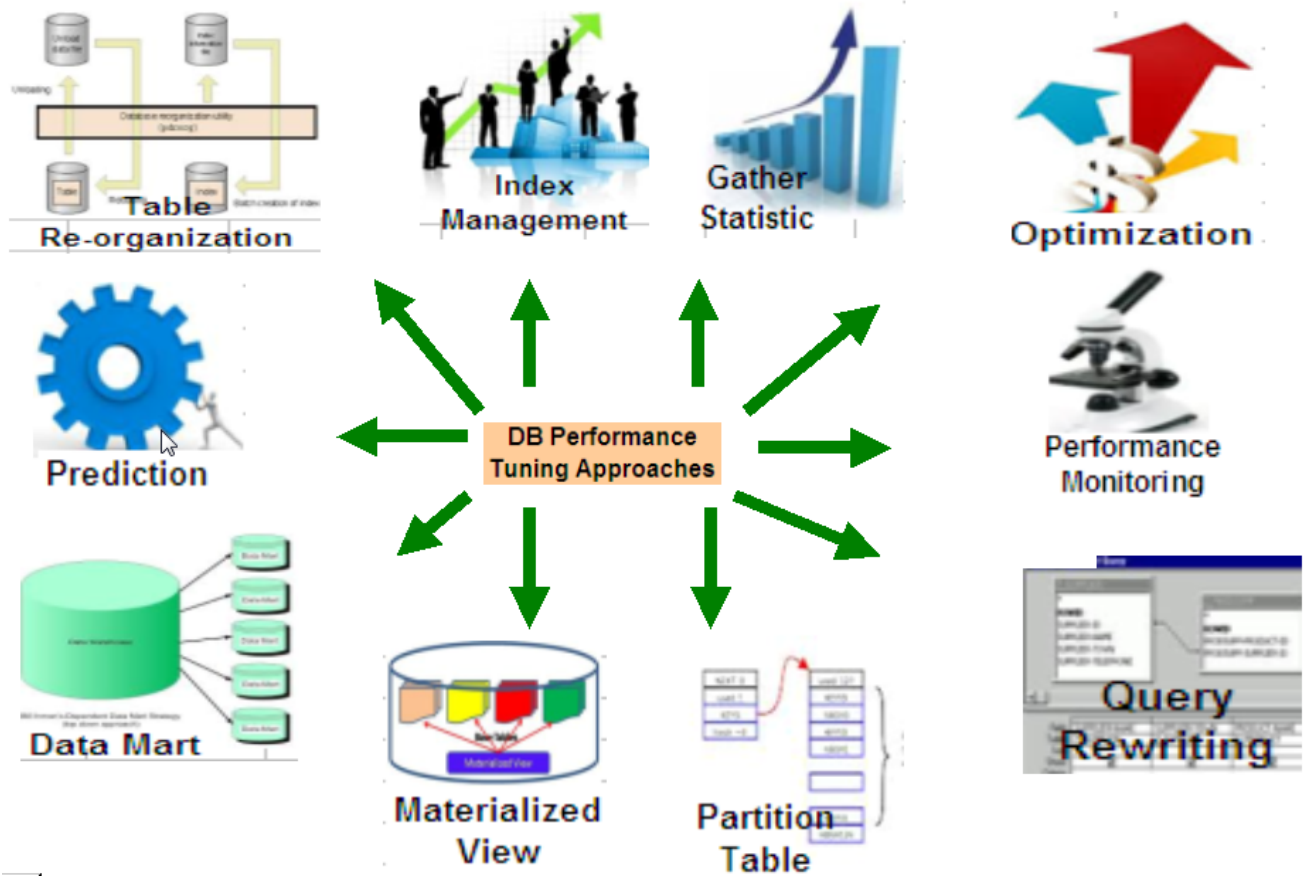

Figure. 2: Silterra's Database Performance Tuning Methods

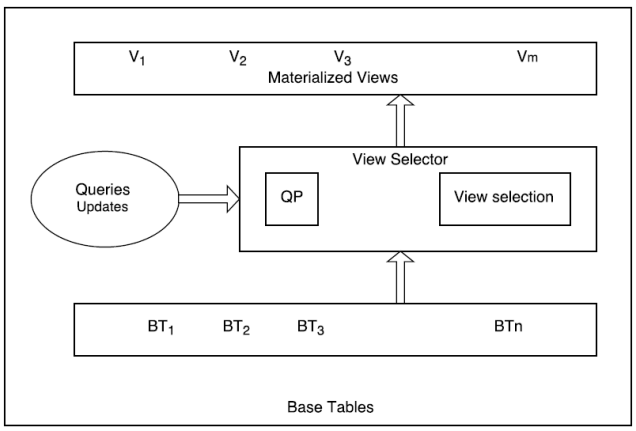

Figure. 3: View Materialization Process (Karde and Thakare,2010) [20] [26]

decision making requirement. Since most of data retrieval from base tables and views are slow, Global Temporary Table (GTT) and MVs are used to speed up data retrieval. These options are good as they able to increase the speed of the slow reports that utilize huge tables. However their usage might need customization. MV acts like a temporary table except that DBAs does not need to do the housekeeping and maintenance of the table. Unlike GTT, MV data will still be available to be used even if the session has ended.

Furthermore, MVs are easier to be administered since they will be auto refreshed at a predefined time. MVs are beneficial in applications such as data warehousing, replication servers, data recording systems, data visualization and mobile systems [21], [22]. Since MVs store the results of the query, it makes MVs readily available when needed to answer the query [18].

Complex SQL query processes such as table joins, selection, aggregation, and calculation have been executed during the predefined time to refresh the MVs. For this reason, SQL query submitted against MVs will be a simple SQL statement. MVs are able to minimize the overall execution time of the workload of queries [20]. The results of the query will be loaded into the storage where these results are more reliable because they only retrieve the required data in the MV. MV has been proven to be an excellent technique in decision support applications, and to preserve the integrated data to ensure better access, performance, and high availability [16].

According to Oracle Manual 2017, SQL query used to create a MV can be categorized into simple and complex [27], as illustrated in Figure 4. The diagram illustrates two databases: the master database (with two base tables) and the MVs database (with two methods for creating MVs). Method A is an example of a complex MV where the join operation is defined during MV creation. During MV refresh, the SQL with join operation will be executed. Since the MV is complex, it performs slower complete refreshes while the simple MVs can perform quicker fast refreshes [27]. Simple

MV is created from one base table and is able to be refreshed quicker as compared to complex MVs. In this example, method B is a simple MV where MV is created using single base table as a source of data. In order to get the same result for this example, a view to join both the MVs is created. The performance of method A will be faster as compared to method B. However the simple MVs can be refreshed more efficiently using fast refresh. These examples show that selection on MV category (either simple or complex) is based on the system's requirements. If the user needs regular refreshes, simple MVs are recommended. However, users can also select complex MV if they need the data faster but the refresh will be less frequent.

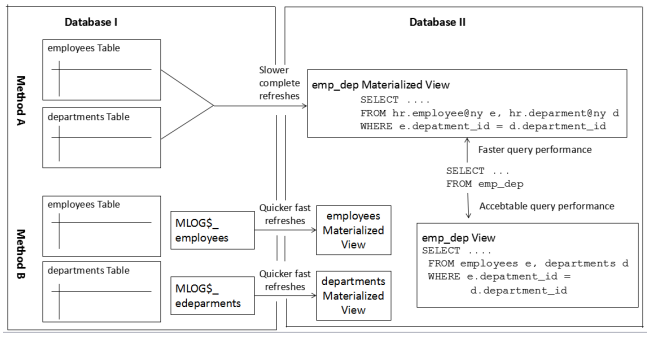

Figure. 4: Comparison of Simple and Complex MVs (Oracle's manual) [27]

Figure 5 shows the structure of how MVs are beneficial in Silterra's MES environment. The historical database (HDB) is a FACTORYworks (FW) module that is stored in a data repository organized for efficient data retrieval and is completely separated from the production FW DB [28]. A utility, called HDB Extract utility, loads the data from FW to HDB in almost real time.

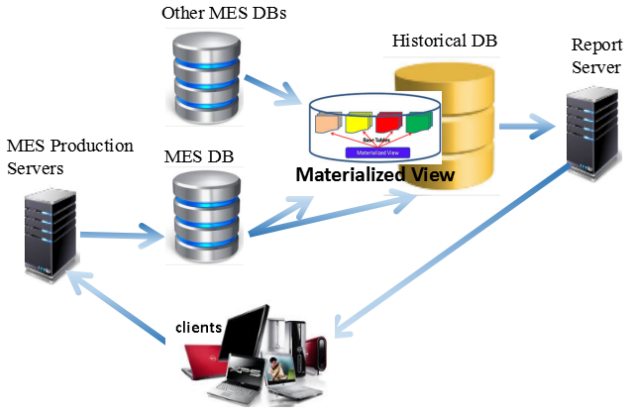

Figure. 5: Implementation of MV in Production Environment

All the reporting utilities need to be run against HDB so that there will be no impact on the performance of production database. However, some information that are not available in HDB are stored in production databases. MV can be used to retrieve the information from production database with lesser impact provided that the MVs are created and executed in HDB. It is able to solve performance issues since reports and queries will select MVs in HDB. With the use MVs, complex joins with aggregation between different MES databases are able to be executed with very less impact. Thus, MVs are useful in the situation where there is increasing number of report request, consistent data growth and highly demand data.

V. Performance Improvement Analysis Using MVs

In order to compare the performance of SQL queries with and without MVs, an experiment has been conducted. The experiment was executed on Fujitsu M5000 with 3CPU of SPARC64 VII 2.6GHz and 32 GB memory on the MES database server that was running on Oracle 11g in the Linux platform. The queries are created using TOAD for Oracle and Oracle SQL Developer. The experiment has been conducted using three MES databases that are: WIP, Historical WIP (HDB), and Equipment Monitoring (EQP) database. Since WIP and EQP databases are riskier to be interrupted, we replicated the production environment. This is to ensure that the query optimizer will select the same execution plan for the problems query [15]. For HDB database, we are able to execute the experiment in the production environment. The risk of performance degrade can be minimized by bounding the impact of the experiment, ensuring inexcessive use of database resource, and maintaining low overhead on the production system [15]. 100 sets of query have been created for the experiment. Each set consist of query to base table (QBT) and query using MVs (QMV). We created a mixed type of SQL queries, involving single to multiple tables (of different databases), with varying result set size. The queries are from simple to complex queries (with aggregation, summation and grouping). To ensure accuracy, the output of the query using QBT and QMV in the same set must be identical. If the results are inconsistent, correction will be made by recreating MVs or QMVs. The average elapsed time for query execution for the query sets are recorded, where the query execution is repeated for three times.

VI. Results and Discussion

Figure 6 shows the result of elapsed time (in seconds) for the query sets under measure. The line chart of elapsed time for 100 set queries has been plotted in QBT elapsed time order. Even though it seems in the chart that performance of QBTs and QMVs are similar for more than half of the query sets (due to large time range in Y axis), QMVs do perform better than QBTs for most cases.

We have conducted the analysis with Performance Improvement (PI), as a way to study a relation and change in a variable. Based on literature, PI has been used to show the percentage improvement difference between two results [30], [15], [31]. While Herodotou *et al.*, (2009) used PI to evaluate the effectiveness of zTuned to find better execution plans for poorly performing queries, LeFevre *et al.*, (2014) used PI to analyse query rewrite using optimistic view for execution time [30], [15]. Tang *et al.*, (2016) used PI to prove that their algorithm for Map Reduce Workloads are 15% to 89% better than the currently unoptimized Hadoop in terms of reduction in running time [31]. These

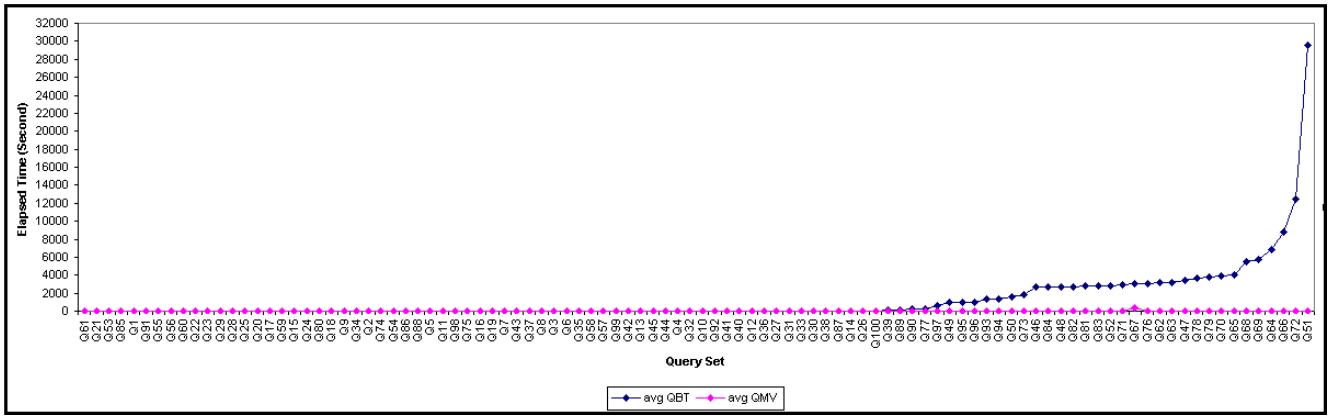

Figure. 6: Experiment Results of QBTs and QMVs in Elapsed Time

examples show that PI is widely used to examine the differences between two numbers effectively in percentage format. PI is calculated by dividing the difference between two numbers by the original number and multiply the value by 100 [15],[31],[29]. Percentage difference equals to the value of the change, divided by the main reference number, all multiplied by 100. In the experiment, PI is calculated using QBT and QMV elapsed time where QBT's elapse time is used as the reference number, as the following:

$$\text{PI} = \frac{QBTElapsedTime - QMVElapsedTime}{QBTElapsedTime} \times 100$$

As smaller elapsed time value is desirable, positive percentage values indicate elapsed time has improved for queries using MVs. As illustrated in Figure 7 there is an average improvement of 82.26% for QMV sets. We observed that 98% query sets of QMV show higher improvement and 48% set of queries show nearly 100% of PI. In particular, for `Q51`, QBT with the highest elapsed time (29573 second) has been tuned to 0.69 second with QMV. In this case, PI is 100%. More than half queries (59) perform very well with PI more than 90%.

The results show that QMVs perform significantly faster as compared to QBT in most query sets as expected. Nevertheless, in this experiment we are more interested to discover QMVs that exhibit low performance, where PI is less than 50% and those with longer elapsed time (more than 10 seconds).

Table 2 highlights the presence of 13 query sets with PI less than 50% and 13 QMVs with more than 10 seconds elapsed time. Even though the overall results show that MVs are able to tune most queries under measure, there are cases where further tuning is needed.

Among the queries that have PI less than 50%, there are two queries namely `Q21` and `Q32` show negative values (-7.27% and -0.32% respectively). This case signifies that, the use of MV is not enough to tune the queries' performance. The possible reasons for this case are: 1) the size of the tables used by this queries are relatively larger as compared to other queries' table size (`Q21`'s table size is 85941.84MB, and `Q32`'s table size is 22607.58MB), and, 2) the way QMV is written (i.e not using the correct index).

Attention must be given for `Q67` as even though PI yielded is high (89.15%), the elapsed time is the highest (with 328 seconds). Other queries that exhibit high PI, but more than 10 seconds elapsed time are `Q83`, `Q79`, `Q71`, `Q76`, `Q68`, `Q69`, and `Q66`. Thus, for applications that aims for speed, PI may not be used as the only factor for consideration in adopting MVs. Furthermore, while offline analysis can reveal the usefulness of MVs for routined-based queries, ways to determine MVs' usefulness for ad-hoc queries will be very much needed in many real-time manufacturing applications today.

Table 2: Experiment Results of QBT and QMV elapsed time order by PI

QuerySets	QBT	QMV (rewrite)	PI(%)
Q21	0.18	0.20	-7.27
Q32	12.72	12.80	-0.60
Q31	35.57	35.37	0.56
Q27	22.39	21.73	2.92
Q40	16.14	13.81	14.47
Q44	11.88	9.80	17.51
Q59	0.93	0.71	23.57
Q2	2.04	1.40	31.65
Q98	3.00	2.00	33.33
Q17	0.93	0.60	35.36
Q37	7.49	4.81	35.80
Q92	14.68	8.31	43.42
Q25	0.84	0.42	49.60
Q100	58.00	16.00	72.41
Q67	**3029.15**	**328.78**	**89.15**
Q83	2795.15	30.13	98.92
Q79	3774.14	24.18	99.36
Q71	2980.77	12.49	99.58
Q76	3064.49	10.03	99.67
Q68	5547.17	11.81	99.79
Q69	5730.87	10.57	99.82
Q66	8832.00	11.77	99.87

VII. Conclusion and Future Work

As a conclusion, to maintain good database performance, effective monitoring is crucial especially in manufacturing industry that deals with massive volumes of data. Nevertheless, monitoring processes through reports generation can be affected by slow SQL queries. In this paper, the common flow of SQL rewriting has been presented, where focus is given on MVs for SQL tuning adopted by Silterra. Materialized views have been useful in optimizing queries for many years. Nevertheless, based on the results of the experiment presented in this paper, there

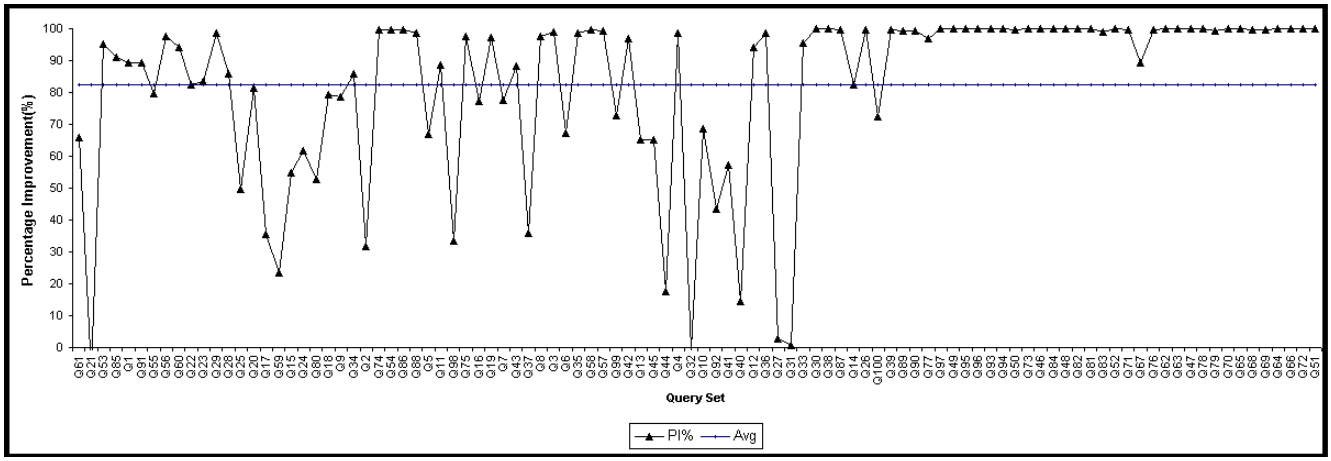

Figure. 7: Experiment Results on Elapsed Time Improvement

are cases that suggests that MVs are inadequate to improve SQL queries' performance. Therefore, further investigation is needed in order to find ways to deal with this limitation.

Acknowledgments

We acknowledge financial assistance received from the Ministry of Education Malaysia under Fundamental Research Grant (FRGS/1/2015/ICT04/FTMK/02/F00289) and the support from Universiti Teknikal Malaysia Melaka.

References

[1] H. Zhou, Z. Yao and H. Xiao, The Design Research on the Storage Structure for Large Amount of Data of the Database, *Proceeding of the International Conference on Advance Mechatronic System*, Zhengzhao. China, pp.384-388, 2011

[2] A. Jacobs, The Pathologies of Big Data, *Communications of the ACM*, v.52 n.8, pp.36–44, 2009

[3] M.A. Chik, S.M. Hazmuni, U. Hashim, Z. Yao and H. Xiao, Industrial Engineering Roles in Semiconductor Fabrication, In *Asia Pacific Industrial Engineering and Management Systems Conference APIEM* , pp.7-10, 2010

[4] K. Ibrahim and M.A. Chik and U. Hashim, Horrendous Capacity Cost of Semiconductor Wafer Manufacturing, In *2014 IEEE International Conference on Semiconductor Electronics (ICSE2014)*, pp.329-331, 2014

[5] T. Ponsignon and L. Monch, Architecture for Simulation-Based Performance Assessment of Planning Approaches in Semiconductor Manufacturing. *In Proceedings of IEEE Winter Simulation Conference, Neubiberg. Germany*, pp.3341-3349, 2010

[6] L. Wein, Scheduling Semiconductor Wafer Fabrication. *IEEE Transactions on Semiconductor Manufacturing*. v1.n.3, pp.115-130, 1988

[7] R.S. Chen, C.M. Sun, M.M. Helms and W.J. Jih, Aligning Information Technology and Business Strategy with a Dynamic Capabilities Perspective: A Longitudinal Study of a Taiwanese Semiconductor Company, In *International Journal of Information Management*, v28 n.5, pp.366-378, 2008

[8] R. Birke, M. Bjoerkqvist, L.Y. Chen, E. Smirni and T. Engbersen, (Big) Data in a Virtualized World: Volume, Velocity, and Variety in Cloud Datacenters, In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*, pp.177-189, 2014

[9] B. Darmawan, G. Groenewald, A. Irving, S. Henrique and M. Snedeker, Database Performance Tuning on AIX, *IBM International Technical Support Organization*, pp.291, 2003

[10] D.K. Burleson, Advanced Oracle SQL Tuning the Definitive Reference, Rampant Tech Press, (2nd ed.), 2010)

[11] T.M. Connolly and C.E. Begg, Database Systems: A Practical Approach to Design, Implementation and Management, SQL: Data Manipulation in A.D. McGettrick (3rd ed.), pp.110-155, 2002

[12] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait and M. Ziauddin, Automatic SQL Tuning in Oracle 10G, In *Proceedings of the Thirtieth International Conference on Very Large Data Bases Volume 30*, VLDB 04, Toronto, Canada,pp1098-1109 ,2004

[13] R. Schumacher, Oracle Performance Troubleshooting, *Oracle Performance Troubleshooting: With Dictionary Internals SQL and Tuning Scripts Oracle In-Focus series*, Rampant Techpress ,2003

[14] M. Ziauddin, D. Das, H. Su, Y. Zhu and K. Yagoub, Optimizer Plan Change Management: Improved Stability and Performance, In *Oracle 11G, Proc. VLDB Endow.*, v.1 n.2, pp.1346-1355, ,2008

[15] H. Herodotou and S. Babu, Automated SQL Tuning through Trial and (Sometimes) Error, In *Proc. of DBTest 09*, ACM, ,2009

[16] P. Bagale and S.R. Joshi, Optimal Materialized View Management in Distributed Environment using Random Walk Approach, *Journal of Advanced College of Engineering and Management*, v.1, pp.67073, 2016

[17] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li and B. Qiu, BigDataBench: A Big Data Benchmark Suite from Internet Services, In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp.488-499, 2014

[18] S. Kurzadkar and A. Bajpayee, Anatomization of Miscellaneous Approaches for Selection and Maintenance of Materialized View, In *IEEE Sponsored 9th International Conference on Intelligent Systems and Control (ISCO)2015*, pp.1-5, 2015

[19] T.F. Yu, T. Tong, M. Xiao and R. Jack, Materialized View Tuning Mechanism and Usability Enhancement, In *Distributed Computing and Internet Technology: Third International Conference*, Heidelberg, Berlin, pp.347-360, 2006)

[20] P.P. Karde and V.M. Thakare, An Efficient Materialized View Selection Approach for Query Processing. *Database Management, Journal of Computer Science*, v.10 n.9, 2010

[21] C. Surajit and D. Umeshwar, An Overview of Data Warehousing and OLAP Technology, *SIGMOD Rec.*, v.26 n.1, New York. USA, pp.65-74, 1997

[22] S. Chen and E.A. Rundensteiner, GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views, In *21st International Conference on Data Engineering (ICDE'05)*, pp.552-563, 2005

[23] A.N.M.B. Rashid and M.S. Islam , An Incremental View Materialization Approach in ORDBMS, In *International Conference on Recent Trends in Information*, Telecommunication and Computing pp. 105-109 ,2010)

[24] P.P. Karde and V.M. Thakare, Selection of Materialized View using Query Optimization in Database Management : An Efficient Methodology, *International Journal of Database Management Systems IJDMS*,4, pp.116-130,2010

[25] H. Gupta, Selection of Views to Materialize in a Data Warehouse, *Proceedings of ICDT*, pp.98-112, 1997

[26] S.R. Valluri and S. Vadapalli and K. Karlapalem, View Relevance Driven Materialized View Selection in Data Warehousing Environment, Aust. Comput. Sci. Commun., *IEEE Computer Society Press*, pp. 187-196, Los Alamitos, CA, USA ,2002

[27] Oracle, Database Administrators Guide Read-Only Materialized View Concepts, Oracle Database 12c Release 2, *Oracle Online Documentation*, pp.70-77 ,2017

[28] FASTech Integration, Inc.,FACTORYworks Historical Database and HDBextract Utility Guide, in *FACTORYworks Version 2.3 and FACTORYworks-Plus Version 2.3*, Fastech Integration, Lincoln North, 1998

[29] Wikipedia, Relative Change and Difference Inc., From Wikipedia, the Free Encyclopedia, https://en.wikipedia.org/wiki/Relative_change_and_ difference

[30] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis and M.J. Carey, Opportunistic Physical Design for Big Data Analytics, In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ACM, pp.851-862, 2014

[31] S. Tang, B.S. Lee and B. He, Dynamic Job Ordering and Slot Configurations for MapReduce Workloads, *IEEE Transactions On Services Computing*, IEEE, 9,pp.4-17, 2016

[32] K. Ibrahim, M.A. Chik, and U. Hashim, Horrendous Capacity Cost of Semiconductor Wafer Manufacturing, In *International Conference of Semiconductor Electronics 2014*, IEEE-ICSE2014 Proc. pp. 345348,2014

Author Biographies

Norazah Md Khushairi, is a Section Manager in Data Management and System Enhancement at Department Computer Integrated Manufacturing (CIM) in Siltera Malaysia Sdn Bhd. She was born in Kedah, Malaysia. She has 18 years experience in Semiconductor Industry

and MES databases. She is proficient in applications architecture, developing MES reporting , web application and data management. She graduated in Bachelors Degree of Computer Science from Universiti Sains Malaysia. Her major field is computer science and with specialization in data management. She has Master in Information Technology, Universiti Utara Malaysia. Currently pursuing PHD in Information Technology with Universiti Teknikal Malaysia. Her research interest in Database Performance and SQL tuning.

Dr. Nurul Akmar Emran was born in Melaka, Malaysia. She received bachelor degree in Management Information System (MIS) from the International Islamic University Malaysia, in 2001, Msc in Internet and Database Systems from London South Bank University in 2003 and Ph.D. degree in computer science from the University of Manchester, UK in 2011. In 2004, she joined the department of Software Engineering, University Teknikal Malaysia Melaka, as a Lecturer, and in 2011 she became a Senior Lecturer. She holds Oracle Certified Profesional (OCP) in 2007 and her teaching mainly covers database-related subjects at undergraduate and postgraduate levels. Her current research interests include storage space optimization, query processing and data quality.

Anil Kumar Menon is a Technology Architect in Oracle Corporation, Malaysia. He was born in Kerala India in 1969. He received bachelor degree in Applied Science Computer Technology from Coimbatore Institute of Technology, Tamil Nadu India in 1991. His major field is in applied sciences and he specializes in computer technology. He has 20 years of experience in IT industry especially in business Intelligence, IT Architecture and Database. He involved in developing IT strategy and architecture for banks, and where he was appointed as advisor for banks' IT architecture and strategy. He also worked on a number of data warehousing projects, content management projects and projects in J2EE. He is a well-versed practitioner in banking and financial services sector.