**Abstract**

Modern, cost-based, relational database performance is reliant upon a number of optimizer statistics, influencing decisions taken by cost-based optimizers, making it a detrimental task to constantly maintain a latest version. Current techniques to ensure updated optimizer meta-data does not factor in underlying, shifting day-to-day workloads; a scheduling puzzle which varies between different environments. This study tackles identification of underlying transactional activity patterns so as to address this scheduling problem through recommendation during when and which statistics are eligible for upkeep.

**Introduction**

Amongst the key processes to ensure efficient execution of queries in a relational database, is the writing of well-crafted query text and the execution of a compiled access plan that aligns with available run time computational resources [15]. Informally referred to as an execution plan, an access plan is a set of steps that constitutes safe, correct and efficient access and retrieval of data from an underlying database. Efficiency is a measure expressed in terms of computational resources parameters (e.g. disk bandwidth, number of CPU cycles) and performance characteristics (e.g. response time). In general, finding the most efficient access plan for a general purpose relational calculus query, on which SQL’s queries were initially built, is a NP-hard problem [1, 2]. Query optimization, that has to cater for multiple and varied categories of database schemas, use statistics related measures [3]. As indicated in the proposal report [15], they are also referred to as optimizer statistics, and are utilized to translate a query into a reasonable access plan. For the query optimizer component of a DBMS to make more informed decisions, statistics are generated based on the volume and distribution of the underlying data. Therefore this relationship ties the effectiveness of an SQL execution plan upon the maintenance and upkeep of optimizer statistics, creating a dependence on maintaining accurate statistics pertaining to the overall system and application performance [4]. Optimizer statistics do not necessarily make query optimization tractable and also comes at a cost; i.e. to generate and hold. The use of statistics in query optimization can generate suboptimal plans and subpar performance in the case of no, insufficient or stale (non-updated) statistics (also referred to as stats here onward), which is often the case when new data is introduced in the underlying schema, as well as drastic and abrupt modifications of existing data, followed by no updating of optimizer stats [4].

**Objectives**

Therefore this research task will attempt to tackle the following challenges:

1. Identification of optimum maintenance windows in the database day to day schedule of lowest database activity, where maintenance windows can be established to generate optimizer statistics without compromising the day to day batch execution.

2. Once a maintenance time window has been established, three challenges can be highlighted:

1. Prioritization of which schema objects (e.g. database tables, columns, indexes, partitions) to gather stats upon. Due to the day to day execution workflow affecting and depending on multiple schema objects for efficient execution of operations, it is detrimental to gather optimizer stats for the correct database structures as required by the impending schedule. This pinpointing of which specific objects to generate stats upon, depends on the flow of execution, during which such a proposed system must be aware of what is going to be executed next so as to be pre-emptive with the stats generation process.
2. Prioritization of which optimizer statistics to gather. With multiple types of statistics, each possessing varied importance subject to the task at hand, the decision making for statistics generation must be intelligent enough to prioritize certain types of statistics over others [14], whilst keeping within the constraints of the day to day schedule.
3. Finally, those prioritized statistics found to be stale, must be generated within the predicted maintenance time window, to avoid exceeding the allocated time for stats generation and risk conflicting with a batch schedule. These stats generation jobs need to be weighted in order of time taken to complete, and otherwise dropped in favor of other maintenance jobs which are predicted to finish in the allocated time, and of more benefit for the overall upcoming schedule.

**Background and Literature Review**

A problem stems from the constant requirement of statistics upkeep, a process which is traditionally overseen by the system’s database administrator (DBA), a task which requires adaptation to schema changes (e.g. new attributes or indexes) and new data to ensure optimum statistics generation for the ensuing database workloads. Many DBMSs have introduced automated processes for the task at hand [5], however these tend to tackle the problem by introducing fixed automated jobs of stats gathering, which are set to run at predetermined points in time as defined by the DBA. Although this alleviates the DBA’s responsibility, it still does not factor in the day to day shifting demands of varied business work flows, a constant and real issue for many online transaction processing (OLTP) and online analytical processing (OLAP) systems [4] whose work schedules are driven by service level agreements (SLA) which must be met and delivered in time. Another facet of this problem is that due to very stringent time schedules of varied database workloads, it is often a problematic task to obtain an optimal time window which can be allocated for database maintenance and optimizer statistic generation, a slot of time where database activity is at its lowest so as to reduce any negative impact on the critical batch and online processing. Past literature [6] suggests ‘just in time’ approaches, where in relevant statistics are gathered prior to query execution. Such a task is considered too workload specific to solve by traditional automatic approaches, and too costly to solve by manual intervention; consequently an ideal scenario for a machine learning approach.

Further to the above, it is highlighted [6, 7, 10] that the issue is not simply a task of determining the occurrence of a maintenance window where in optimizer statistics can be gathered. It is also the quality of the gathered stats which determines the efficiency of an optimizer stats gathering processes, both in terms of the stats gathering job itself, and also the degree of accuracy with which it can anticipate subsequent reliance of stats in the future. Attempts have been made to tackle this fundamental aspect of statistics upkeep through automated means [3], particularly through use of novel techniques which aim to reduce the varied amounts of stats being gathered, and pinpointing those that are needed. Other techniques involve the usage of rule defined solutions, as demonstrated on DB2 [7], where in it is recognized the need to offload the task of maintenance upkeep from the DBA, and offload this work to be done automatically by the database itself. Through usage of DB2’s RUNSTATS statistics collection utility [8, 9] and its constant statistic monitoring, this technique prioritizes which schema tables are eligible for statistics upkeep. Whilst this technique approximates an automated routine which caters of statistics upkeep, it does not take into consideration two important factors:

1. The routine for statistics generation is executed in a predefined maintenance window, specifically set by the DBA. In addition, the aforementioned literature [7] proposes a throttling mechanism of stats generation during the day, risking the potentiality that stats upkeep becomes a lengthy process, whilst putting a tamper on available hardware resources due to the constant background stats process.
2. Although the proposed solution attempts to achieve schema object priority for statistic generation, it does so only from the perspective of database tables, giving no attention and purpose to index and column statistics, which are equally relevant to the proposed optimization problem.

**Methodology – Data Acquisition**

Therefore to address the previous afore mentioned objectives and effectively produce a number of solutions worthy of satisfying the proposed challenges, a representative dataset is required to base any testing and evaluation upon. A decision support TPC Benchmark schema [11, 12] has been chosen, particularly for its many varied SQL transactions, large quantities of data capable of generating, and its open source nature. The TPC schema will be installed upon an Oracle12c [13] instance, which will serve as the foundation for future testing. The decision to base the study upon an oracle instance in contrast to otherwise using other versions of Oracle DBMSs, or other non-oracle DBMSs, takes the following into consideration:

1. The final proposed solution/s should be capable of generalizing, irrespective of underlying relational database vendor, or database version.
2. The level of information capture offered by the technology in relation to runtime execution.
3. No changes present which pertain to the proposed objectives, between Oracle 12c and Oracle 18c versions.
4. Familiarity with Oracle relational database usage, in contrast to other relational database technologies.

As a test bed for future evaluation, a total of three schemas are created and loaded with the TPC-DS benchmark suite, sized as 1G, 10G, and 100G (Gigabytes) respectively. Following this, a thorough benchmark of all TPC provided SQL transactions will be carried out (with / without optimizer statistics presence), so as to identify the top consumers and measure the performance degradation with respect to optimizer stats presence per schema.

To simulate an active working database schedule, a number of simulated database schedules will be executed and monitored upon the already established volumes, built using the provided TPC-DS transactions. Each schedule would require to execute with varying degrees of transaction loads, underlying data volumes, rapidness of transaction execution, and more. During each schedule run, a number of metrics pertaining to the infrastructure behavior and overall transaction load will be extracted, for which will be used for the next phase of the experiment, as highlighted below. The type of metrics eligible for extraction will pertain to each respective SQL behavior and respective resource consumption. A brief, summarized outlook of the type of metrics which will be captured, proceed as follows:

* CPU consumption (CPU cycles under work)
* Number of physical data block reads/writes
* Number of data buffer hits
* Result cache usage
* Total elapsed time

Effort has also been made so as to collaborate with industry partners and acquire a similar workload/schedule dataset, which would portray a real-live working environment in contrast to a TPC synthesized schema. Due to respective company data policies and University of Malta submission regulations, this option is being considered second after the publicly available TPC benchmark suite, prioritizing testing and evaluation on the synthesized data.

**Methodology – Machine Learning Approach**

Using the pre-stored schedule metrics, attempts will be made to incorporate a series of machine learning methodologies upon the acquired and already stored scheduling metrics. These metrics will serve a two-fold purpose:

1. Incorporation of a number of regression / clustering techniques which can be applied to determine when during the day, a particular database instance is at its lowest, and highest activity, allowing better accuracy pertaining to ideal timeslots when optimizer statistics processes can be carried out.
2. Identification of SQL top consumers, and SQL explain plan supervised learning, allowing the drill down and identification of what makes an SQL execution so costly on the database, and attempting to lower this cost through optimizer statistics generation. The choice of which optimizer statistics, and which schema objects are most susceptible to improve from optimizer stats, should also be influenced through machine learned techniques.

**Evaluation**

The evaluation process will be categorized and distributed over a number of categories. Initially, each TPC-DS transaction will be executed across varied volumes, with and without optimizer statistics, to establish the detrimental performance of optimization stats upon the TPC-DS schema.

After executing a number of varied workload schedules against the pre-established TPC-DS schemas, the before mentioned clustering and regression techniques need to be evaluated for their accuracy and precision with which a database window is categorized into activity windows. The effectiveness of these techniques can be measured through how accurately they categorize future timelines, in terms of database activity (percentile usage). This is achieved by comparing predicted upcoming transactional activity, to actual database workload activity.

Upon establishment of when optimizer statistics are to be run, the proposed solution should be capable of giving recommendations as to which stats should be gathered. The effectiveness of these techniques require the transactional processes to be executed individually, particularly focusing on TPC-DS top consumers, and executing these in a way so as to measure respective time taken, and relevant execution metrics (explain and execution plan results) after optimizer statistic generation, as proposed by the solution.

**Conclusion**

With a degree of coverage pertaining to the useful and more efficient gathering of optimizer statistics in relational databases, current literature leaves room to explore machine learned automated approaches in contrast to current rigid techniques. By carrying out work on the publically recognized TPC synthesized benchmark tools, attempts will be made to mimic realistic transactional workloads, and propose a number of machine learning based methods as to how the optimizer statistics scheduling process can be enhanced.

**References**

[1] A. Kumar and M. Stonebraker, "The effect of join selectives on optimal nesting order", ACM SIGMOD Record, vol. 16, no. 1, pp. 28-41, 1987.

[2] T. Ibaraki and T. Kameda, "On the optimal nesting order for computing N-relational joins", ACM Transactions on Database Systems, vol. 9, no. 3, pp. 482-502, 1984.

[3] S. Chaudhuri and V. Narasayya, "Automating statistics management for query optimizers", IEEE Transactions on Knowledge and Data Engineering, vol. 13, no. 1, pp. 7-20, 2001.

[4] R. Borovica-Gajic, S. Idreos, A. ilamaki, M. Zukowski and C. Fraser, "Smooth Scan: Statistics -Oblivious Access Paths", IEEE, Seoul, South Korea, 2015

[5] D. K. Burleson, Oracle Tuning, The Definitive Reference Second Edition, 2nd ed. Ora, 2008, pp. 241 -280.

[6] N. Bruno and S. Chaudhuri, "Exploiting Statistics on Query Expressions for Optimization", SIGMOD '02 Proceedings of the 2002 ACM SIGMOD international conference on Management of data , vol. 1, pp. 263-274, 2002.

[7] A. Aboulnaga, P. Haas, M. Kandil, S. Lightstone, G. Lohman, V. Markl, I. Popivanov and V. Raman, "Automated statistics collection in DB2 UDB", Proceedings of the Thirtieth international conference on Very large data bases, vol. 30, no. 04, pp. 1158-1169, 2004.

[8] International Business Machines Corporation, Armonk, N.Y., "SYSTEM FOR DATASTRUCTURE LOADING WITH CONCURRENT STATISTICAL ANALYSIS", 5,873,091, 1997.

[9] [3]"IBM Knowledge Center", Ibm.com, 2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSEPEK\_10.0.0/ugref/src/tpc/db2z\_utl\_runstats.html. [Accessed: 17- Aug- 2018].

[10] M. Ziauddin and F. Calif, "METHODS FOR COLLECTING QUERY Attorney, Agent, or Firm Wagner, Murabito & Hao LLP WORKLOAD BASED STATISTICS ON COLUMN GROUPS IDENTIFIED BY RDBMS OPTIMIZER", 6,029,163, 2000.

[11] <http://www.tpc.org/tpcds/>

[12] http://www.tpc.org/tpc\_documents\_current\_versions/pdf/tpc-ds\_v2.9.0.pdf

[13] http://www.oracle.com/us/corporate/features/database-12c/index.html

[14] Best Practices for Gathering Optimizer Statistics", Oracle White Paper, 2012

[15] Proposal Report