Intelligent Optimizer Statistic Generation for Relational Databases

Gabriel Sammut
University of Malta
gabriel.sammut.13@um.edu.mt

ABSTRACT

Modern relational databases operate through usage of a cost based optimizer (CBO), whose task is to generate efficient access paths for compiled structured query language (SQL). Access paths taken by the optimizer are influenced by a number of optimizer statistics. By establishing regular maintenance windows, a database scheduler regularly generates and renews additional and past statistics respectively, in an effort to ensure that any decisions taken by the CBO are based off the latest gathered information. These maintenance windows are often scheduled in a fixed manner, with DBA established points in time, with little to no indication as to when and which statistics ought to be gathered or renewed. This approach is often not enough to satisfy the optimizer's needs, leading to skewed access plans and performance degradation. This study tackles identification of underlying transactional activity patterns, aiming to address this scheduling problem through intelligent based recommendations for statistics upkeep and maintenance.

1. 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 [31]. 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 SQLs queries were initially built, is a NP-hard problem [18, 13]. Query optimization, that has to cater for multiple and varied categories of database schema architectures, uses statistics related measures to generate efficient data retrieval methods [7].

As previously indicated [31], they are also referred to as optimizer statistics, and are utilized in the process to help 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 dependency on maintaining accurate statistics pertaining to the overall system and application performance [30]. 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 sub par 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 [30].

Traditional and common used approaches to ensure statistical upkeep resort to hard set, predefined maintenance windows, often defined by manual DBA intervention. The decisions pertaining to which schema objects are eligible for renewal are determined through static database values which determines the threshold limit until a database object is eligible for stats renewal. This approach presents a rigid approach towards maintaining updated optimizer statistics, consequently playing an adverse affect on execution performance of the business dictated transactional workload. This reliance on human intervention, and rigidness to the statistical collection model, exposes a potential pitfall - one which can be potentially addressed through artificially learned techniques, making the RDBMS reliant on itself to better and more accurately address this conundrum.

2. OBJECTIVES

Therefore this research task will attempt to tackle the following challenges by providing justifiable resolutions to them, through a number of machine learning techniques:

- 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.

- Once a maintenance time window has been established, three related challenges can be highlighted:

- Prioritization of which schema objects (e.g. database tables, columns, indexes, partitions) to gather stats upon. Due to the day to day execution work flow 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 preemptive with the stats generation process.
- 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 [2], whilst keeping within the constraints of the day to day schedule.
- 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.

3. BACKGROUND AND LITERATURE REVIEW

As the demands for data sizing have increased drastically in modern day, data housing, so has too the demand for more efficient data retrieval techniques [28]. With the relational database a well established back bone of innumerable data housing infrastructures, the need for efficient, fast data manipulation (read/write) has become a crucial component of any RDBMS distribution [1, 13, 18, 30].

3.1 Background Domain

At the core of every self contained relational database is an optimizer, whose job is to generate least cost access plans for executed SQL transactions. An access plan is constituted as a set of instructions describing how data is retrieved from the underlying schema, where cost is a measure of invoked computational resources (CPU Cycles, IO Read/Write, etc..) and time required for an SQL's execution (response time) [34]. It should be noted that there exists two types of optimizer variations

- **Rule-based optimizers (RBO)** are a set of self contained rules which inhibit guidelines for the underlying database engine - in an effort to deliver data in the least possible time. Variations of this type of optimizer were prevalent in many database RDBMs. However, a major shift towards cost based optimizers has taken place - the latter cost based version considered to be more elastic than their rule-based counterpart [28].
- **Cost-based optimizers (CBO)** use acquired statistics so as to determine the most efficient access path on the underlying database schema. Through usage of this acquired 'meta-data', the CBO attempts to estimate different variations of SQL access paths, and base decisions on whichever is least expensive [11].

The nature of the proposed solution and reviewed literature concerns itself respectively towards cost-based optimizers, particularly due to the heavy shift towards cost based evaluation throughout the last decade, and also due to the fact that rule base optimizers are not reliant on optimizer statistics. Therefore, RBO access plan formulation and over all accuracy when it comes to rule based decisions, fall outside the remit of this study.

3.2 Optimizer Statistics

On the most abstract level, optimizer statistics can be described as a collection of data which describes the database and it's respective underlying objects [3]. The type of information gathered by particular vendor specific RDBMS varies, however it is staple of such technologies to take into account a number of statistical types. Often, these statistical categories revolve around the base calculation of table cardinality estimates, counting of tuple sizes and amounts, number of underlying data blocks constituting schema objects, data skewness representations, clustering factors, and more [5, 3, 29, 19]. Oracle RDBMS distributions categorize statistics into four groupings, which adequately describe the resources and underlying data distribution available which are visible to the optimizer upon SQL compilation: [3]:

- **Table Statistics** Includes information pertaining to table number of rows, underlying table data blocks and average row length. These statistics are kept on a per table basis, and are used in conjunction to other optimizer statistics so as to compute various operations in an access plan.
- **Column Statistics** These type of statistics pertain to columnar information, incorporating cardinality estimates on a per column basis, min/max column values and otherwise. They are used in conjunction with table statistics to help the optimizer formulate an efficient access plan. An additional type of statistics can be added in the form of histograms, so as to better represent data skewness [23].
- **Index Statistics** This variety of statistics provides information pertaining to schema held indexes, particularly denoting counts of index distinct values, index depth, index clustering factors and leaf block counts. This type of information often helps the CBO decide between opting for a full table scan, or choosing data scanning and retrieval through an index.
- **System Statistics** These type of optimizer statistics influence the CBO to make more informed decisions based of the underlying hardware distribution [25]. They contain information pertaining to available CPU speed, IO seek time, number of available cores, and more. In contrast to the other statistic types, system statistics only need to be gathered a number of finite times, often when significant hardware changes are introduced.

3.3 Problem Domain

Firstly, it should be noted that the act of statistics gathering and query optimization is a decoupled process [11]. The statistics gathering module has limited insight to what transactional load is present on the underlying database. Daily queries and database jobs are not considered when it comes to optimizer statistic upkeep, taking into consideration only a percentile delta worth of data changes.

Additionally, a problem stems from the constant requirement of statistics upkeep, a process which is traditionally overseen by the systems database administrator (DBA), a task which requires adaptation to schema changes (e.g. new attributes or indexes) and new data, so as to ensure optimum statistics generation for the ensuing database workloads. Varied DBMS distributions have introduced automated processes for the task at hand [10, 15, 29], 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 DBAs 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 on-line transaction processing (OLTP) and on-line analytical processing (OLAP) systems [30] 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 on-line processing.

Finally, it should also be mentioned that optimizer statistics are gathered based on a rule defined basis, a percentile factor which dictates how stale respectively gathered statistics are on the database schema. Statistics found to be old and invalidated, are considered to be 'stale', and are generally a common culprit to skewed access plans generated by the cost optimizer. [4, 9]. The deciding factor dictating which schema objects are entitled for statistical renewal, varies between one DBMS vendor to the next, often taking the form of a quantitative measure which measures how much data has changed within a schema object. Due to the different varieties of data volumes and data structures in a schema, and the vast gaps between one particular schema design and another, it is an impossible task to satisfy this rule with a fit all threshold value. This challenge makes it an ideal scenario for a machine learning approach, which can tailor this value to the schema's needs, on a per schema basis.

3.4 Statistical Timeliness

Prior literature [5] proposes the usage of additional statistical techniques, which aim to utilize an additional small subset of optimizer statistics through simple maintenance of basic table stats. Introducing the concept of SITs (Statistics on Intermediate Tables), this type of meta data is used to accurately model tuple distribution over intermediate nodes in a query access plan. The concerned literature [5] states that this type of gathered statistics improves optimizer accuracy when it comes to efficient generation of SQL access plans, claiming the potential of improving execution timings ten-fold.

Other approaches [4] suggests the use of a throttling mechanism, were by the optimizer statistic generation job is offloaded as a background process which is constantly active. This alleviates personnel responsibility, allowing the database to gather statistics unimpeded at it's leisure, regardless of the underlying transactional activity and job schedule. This however risks long running executions of stats gathering, due to the limited resource pool allocated for such jobs. Another point which can be made in opposition of such a technique, is that concurrent generation of new access plans will be influenced by older/stale optimizer statistics (leading to incorrectly based assumptions and possible access plan skew), in turn leading to process timing degradation and potential missing of SLA cutoffs. In addition, such a technique proposes a resource cap, since adequate computational resources would need to be reserved for statistical analysis and generation, depending on the granularity of the job - an assumption that does not always hold true, depending on the underlying hardware infrastructure.

In contrast to this approach, particular RDBMS technologies allow the usage of 'just-in-time' techniques [21, 12], utilized to influence the optimizer's behavior during compilation of an SQL transaction. Particularly Oracle RDBMS [6] utilize the concept of 'Dynamic Sampling' [2, 3], allowing the collection of additional transactional query schema object statistics during the optimization phase of an SQL compilation. Different levels of granularity can be opted for during this step, each requiring a significant resource and time overhead for this information to be gathered during run time. Dynamic sampling serves to support and enhance already established statistics, so it should be stated that this technique is not an alternative to traditional optimizer statistics [3], as paraphrased: "The statistics gathered in this case are not of high quality or as holistic as the statistics gathered using Oracle's DBMS_STATS package". Although this type of statistics gathering is incurred at run time and therefore has a better opportunity at handling and dictating what computational resources are require to be aligned for a particular transactional computation, dynamic sampling still requires a solid foundation of underlying optimizer statistics to be truly effective.

3.5 Statistical Eligibility

It is also highlighted [5, 4, 20] 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 [7], 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 [4], 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 DB2s RUNSTATS statistics collection utility [17, 14] and its constant optimizer statistic monitoring, this technique prioritizes which schema tables are eligible for statistics upkeep. Whilst this technique approximates an auto-

mated routine which caters of statistics upkeep, it does not take into consideration two important factors:

- The routine for statistics generation is executed in a predefined maintenance window, specifically set by the DBA. In addition, the aforementioned literature [4] 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.
- 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.

4. METHODOLOGY

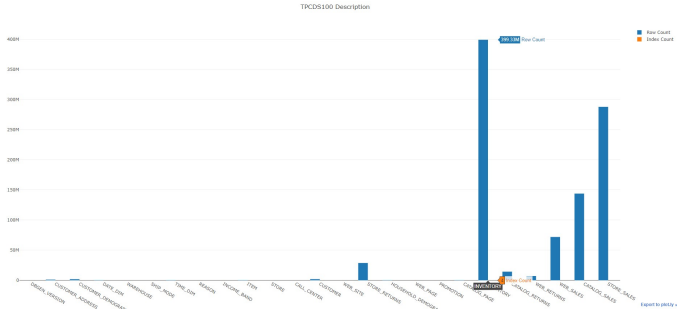
4.1 Data Acquisition

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 [32, 33] has been chosen, particularly for its many varied SQL transactions, capable of generating large quantities of data, and its open source nature. The TPC schema will be installed upon an Oracle12c [24] instance, which will serve as the foundation for future testing. The decision to base the study upon an Oracle instance in contrast to other versions of Oracle DBMSs, or other vendors entirely, takes the following into consideration:

- The final proposed solution should be capable of generalizing, irrespective of underlying relational database vendor, or database version.
- The level of information capture offered by the technology in relation to runtime execution. Oracle offers a number of internal detailed views [22], detailing the resource usage pertaining to underlying database activity.
- The latest version of Oracle RDBMS technologies, Oracle18c, was unreleased during time of consideration. No changes are present which pertain to the proposed objectives, between Oracle 12c and Oracle 18c versions [24].
- 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. The usage of flashback functionality [16] will allow efficient reversal of the transactional workloads so as to ensure that each workload process is based upon the same baseline.

Figure 1: TPC-DS 100 Gigabyte Distribution



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 are 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

Further to the above captured information, AWR and ASH based reports are also being considered, due to their data mining potential in retrieving data pertaining to query performance [27, 26].

4.2 Industry Collaboration

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.

4.3 Machine Learning Approach

Using the pre-stored scheduling metrics, attempts will be made to incorporate a series of machine learning heuristics upon the acquired and already stored scheduling metrics:

- 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.

- 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. Day to day access plans will be monitored for the top most expensive queries, and any inconsistencies will be flagged pertaining to each respective SQL plan in advance, enabling a preemptive performance tuning process to occur before the transaction query has itself started processing.

- The choice of which optimizer statistics eligible for generation / updating should be incorporated in these artificially learned heuristics, enabling the relational database to take better informed decisions as to which type of optimizer stats are opted for.

- The choice of which schema objects are most susceptible to improve from optimizer stats, should also be influenced through machine learned techniques. Due to different schema naming conventions and object types between different database environments, it is difficult to apply a rule based and static solution to this identification process, making it a likely candidate for artificially learned heuristics.

5. EVALUATION

The evaluation process will be categorized and distributed over a number of categories. It should be noted that it is not the impact of optimizer statistics effectiveness which will be measured, a research domain for which vast literature already exists. Instead, due to the nature of the proposed research topic, effort will be taken to gauge the accuracy as to when optimizer statistics are generated upon the database. It is the timeliness and accuracy of the scheduling time slot which will be evaluated, gauging how self reliant the proposed solution is when it comes to basing decisions based off already established transactional execution timings. Factors of time taken to generate new stats, dependency upon upcoming schedule workloads, and relevance of which optimizer related statistics to be gathered, will be considered.

5.1 TPC-DS Top Consumer Identification

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. This will also serve to highlight which transactions are most heavy on the underlying data volumes, allowing future work and evaluation to cater for particular attention on these use cases. It should be highlighted at this stage, that due to particular complex permutations of data access retrieval methods, particularly at the level of limited or no relevant optimizer statistics available, it is expected that particular transactional runs will take a measure of time to complete, orders of magnitude timed across hours or days. For such scenarios, such timings will be time capped - timed out ahead of time and flagged as not being able to complete within formal expectation.

5.2 Scheduling Timeliness

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 low activity database windows are correctly identified. The effectiveness of these techniques can be measured through F-Score, Precision and Recall scorings, as to how accurately they categorize future time lines, in terms of database activity (percentile usage). One of the proposed techniques of evaluation at this phase is to split the 24x7 time window into five (an arbitrary value) minute 'pockets'. Each pocket of time will be flagged with the level of activity established/predicted to be on the database. This technique allows a clear and well understood representation, with which time base scheduling allocation can be verified, allowing the upcoming evaluation techniques to work upon the established assumption that stat generation will be taking place in time slot of low database activity.

5.3 Detecting Access Plan Inconsistency

Upon establishment of when optimizer statistics are to be run, the proposed solution should be capable of giving recommendations as to which transactional workloads should be analyzed for optimizer statistics generation. Through generation of Oracle's PLAN_TABLE [28], predictions can be made as to which access plan the optimizer proposed to take for an SQL execution. It is being proposed at this stage, to have individually trained heuristic models per top consumer transaction, whose purpose is to evaluate changes between daily taken access plans. This enables the solution to flag any access plan inconsistencies, allowing the next phases to take effect. What this essentially means, is that based on the optimizer statistics currently gathered in the data dictionary available to the optimizer, inconsistencies to the usual access plan pattern have been discovered. By flagging such inconsistencies, this enables the solution to further expose which transactional executions require statistical generation and/or upkeep.

Access plan inconsistencies can be injected through usage of optimizer hints [8], and evaluated across each SQL trained model to check whether self induced inconsistencies get flagged. The effectiveness of these techniques require the transactional processes to be executed individually, particularly focusing on TPC-DS top consumers due to the vast amounts of TPC-DS queries which are available.

5.4 Optimizer Statistic Prioritization

With a level of flagged SQL transactional consumers, the next evaluation step requires to gauge the effectiveness as to the decisions taken behind which optimizer statistics are considered eligible for execution. In addition to this and as already stated previously, the quality of the optimizer statistics generation process does not only depend on the type of stats which are renewed, but also which schema objects will considered for this renewal process. Due to the varied permutations for which a cost based optimizer might propose different access plans, it is a difficult task to generate and compare the effectiveness of all possible query plans. In addition, the usage of flashback techniques will be required here, to revert any statistic generation jobs back to an original state, for the purposes of a streamlined and equal based evaluation. Therefore, a number of clearly stated assumptions will be taken when it comes to the evaluation of this step, the likes of which pertain as to the decisions taken behind the statistics prioritization process, and the considered subsets of stats/schema objects which will be considered.

6. CONCLUSION

With a degree of coverage pertaining to the efficient gathering of optimizer statistics in relational databases, current literature leaves room to explore machine learned automated approaches in contrast to current rigid techniques opted for by a range of DBMS vendors. By carrying out work on the publicly 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. A particular focus will be taken to intelligently automate this task, through analysis of past execution activity. This analysis will not only take into account the time activity of the schedule, but also which types of access plans are predicted to change in upcoming executions. This allows the proposed system to identify potential performance degradation ahead of time, and alleviate this through generation of optimizer statistics. These techniques together allow the RDBMS to intelligently schedule a task detrimental to performance, which would otherwise be bound by hard set parameters.

7. REFERENCES

[1] Automatic SQL Tuning in Oracle 10g.

[2] Best Practices for Gathering Optimizer Statistics. 2012.

[3] Understanding Optimizer Statistics. 2012.

[4] A. Aboulnaga, P. Haas, M. Kandil, S. Lightstone, . G. Lohman, V. Markl, I. Popivanov, and . V. Raman. Automated Statistics Collection in DB2 UDB. Technical report.

[5] N. Bruno and S. Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data - SIGMOD '02*, 2002.

[6] Burleson Consulting. Inside Oracle Dynamic Sampling.

[7] S. Chaudhuri and V. Narasayya. Automating statistics management for query optimizers. In *IEEE Transactions on Knowledge and Data Engineering*, 2001.

[8] Daniel Manhung Wong and Chon Hei Lei. Database Object 102N Policy Function 152 (User Registerable) "Policy type specifies When database Server evaluates policy function Or Database Object 104 retrieves previously Policy Function 162 Computed result Memory 172 Memory 174, 2004.

[9] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood. Automatic Performance Diagnosis and Tuning in Oracle. Automatic Performance Diagnosis and Tuning in Oracle. 2005.

[10] Donald K. Burleson. *Oracle Tuning: The Definitive Reference Book 32 of Oracle in-Focus Series.* Rampant TechPress, 2010, Kittrell, Nort Carolina, second edition edition, 2010.

[11] A. El-Helw, I. F. Ilyas, W. Lau, V. Markl, and C. Zuzarte. Collecting and maintaining just-in-time statistics. In *Proceedings - International Conference on Data Engineering*, 2007.

[12] C. N. Hsu and C. A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[13] T. Ibaraki and T. Kameda. On the optimal nesting order for computing N-relational joins. *ACM Transactions on Database Systems*, 1984.

[14] IBM Knowledge Center. RUNSTATS.

[15] I. F. Ilyas, V. Markl, P. J. Haas, P. G. Brown, and A. Aboulnaga. CORDS: Automatic Generation of Correlation Statistics in DB2. Technical report.

[16] J. William Lee, Juan Loaiza, Michael J. Stewart, Wei-Ming Hu, and J. William H. Bridge. United States Patent, 2007.

[17] John Marland Garth, Koshy John, James Alan Ruddy, David Ray Schwartz, and Bryan Frederick Smith. SYSTEM FOR DATASTRUCTURE LOADING WITH CONCURRENT STATISTICAL ANALYSIS, 1999.

[18] A. Kumar and M. Stonebraker. THE EFFECT OF JOIN SELECTIVITIES ON OPTIMAL NESTING ORDER. Technical report.

[19] J. Lewis. Cost-Based Oracle Fundamentals. Technical report, 2006.

[20] M. Ziauddin and F. Calif. Methods for collecting query workload based statistics on column groups identified by RDBMS optimizer, 2000.

[21] Nigel Bayliss. Dynamic sampling and its impact on the Optimizer, 2010.

[22] Oracle. Dynamic Performance (V$) Views.

[23] Oracle. Histograms.

[24] Oracle. Oracle Database 12c: Plug Into the Cloud.

[25] Oracle. System Statistics.

[26] Oracle-Base. Active Session History (ASH).

[27] Oracle-Base. Automatic Workload Repository (AWR) in Oracle Database 10g.

[28] R. Pacholewicz. ASSESSMENT OF IMPACT OF SELECTED FACTORS ON THE EFFECTIVENESS OF COST-BASED OPTIMIZER IN DATABASE SYSTEMS. Technical report.

[29] PostgreSQL. The Statistics Collector.

[30] R. Borovica-Gajic, S. Idreos, A. ilamaki, M.Zukowski, and C. Fraser. Smooth Scan Statistics-Oblivious Access Paths. *IEEE 31st International Conference on Data Engineering*, 31:2–5, 2015.

[31] Sammut Gabriel. Faculty of ICT Master of Science in Artificial Intelligence (Part-Time) Proposal Form Title: Intelligent Optimizer Statistic Generation for Relational Databases. Technical report, 2018.

[32] TPC. TPC - Benchmarks.

[33] Transaction Processing Performance Council. TPC BENCHMARK DS, 2018.

[34] J. Zahir and A. El Qadi. A Recommendation System for Execution Plans Using Machine Learning. *Mathematical and Computational Applications*, 2016.