Summer 2010

# A Framework for the Automatic Physical Configuration and Tuning of a Mysql Community Server

Kevin Spillane
*Regis University*

**Regis University**
College for Professional Studies Graduate Programs
**Final Project/Thesis**

# Disclaimer

**Abstract**

Manual physical configuration and tuning of database servers, is a complicated task requiring a high level of expertise. Database administrators must consider numerous possibilities, to determine a candidate configuration for implementation. In recent times database vendors have responded to this problem, providing solutions which can automatically configure and tune their products. Poor configuration choices, resulting in performance degradation commonplace in manual configurations, have been significantly reduced in these solutions. However, no such solution exists for MySQL Community Server. This thesis, proposes a novel framework for automatically tuning a MySQL Community Server. A first iteration of the framework has been built and is presented in this paper together with its performance measurements.

**Acknowledgements**

I would like to thank my thesis advisor, Mr. Darl Kuhn for his for his guidance and advice, it is very much appreciated.

I would like to thank the participants to the questionnaire research.

I would also like to thank my sister Margaret.

Finally, I would especially like to thank my wife Caroline, for her unfailing love, patience and understanding through many years of study.

In loving memory of my mother, Carmel.

## Table of Contents

Abstract ..... 1
Acknowledgements ..... 2
Table of Contents ..... 3
List of Figures ..... 7
List of Tables ..... 8
Chapter 1- Introduction ..... 9
Statement of the Problem ..... 9
Study Background ..... 9
Research Questions ..... 12
Research Hypothesis ..... 13
Importance of the topic ..... 13
Thesis Structure ..... 15
Scope ..... 15
Success Criteria ..... 16
Chapter 2 – Literature Review ..... 18
Introduction ..... 18
Factors which cause poor performance in relational databases ..... 20
Problem Diagnosis ..... 21
Tuning methodologies ..... 23
Online versus Off-Line Tuning ..... 26
OLTP versus DSS ..... 27
Selection of Digital Tuning Advisors ..... 27
The addition of Physical Hardware ..... 28
Indexing Selection Strategies ..... 29
Memory Allocation ..... 33
MySQL Database Monitoring and Tuning ..... 36
MySQL Cache Memory Tuning ..... 37
MySQL Tuning ..... 40

Chapter Summary ..... 42
Chapter 3 – Methodology ..... 44
Introduction ..... 44
Research Design ..... 44
Suggestion Phase. ..... 44
Design Phase. ..... 47
Evaluation Phase. ..... 47
Hypothesis Testing ..... 48
SDLC Process and Procedures ..... 48
SDLC Key Deliverables ..... 50
Resource Requirements ..... 51
Chapter Summary ..... 52
Chapter 4 – Project Analysis and Results ..... 54
Introduction ..... 54
Review of Existing Solutions ..... 54
MySQLTuner.pl. ..... 55
tuning_primer.sh. ..... 57
Results of Analysis. ..... 58
Questionnaire Results ..... 59
Interest Level in proposed product. ..... 59
Experience of Participants. ..... 60
Required Feature Set. ..... 60
Index Tuning Advisor Usage. ..... 62
Cache Tuning Advisor Usage. ..... 62
Usage levels of the proposed tool. ..... 63
Suggestions for additional features. ..... 63
Experience of other Automated Database Tuning Tools. ..... 64
Current Method of Tuning MySQL Community Databases. ..... 65
Comments and suggestions. ..... 65
Elaboration Phase Deliverables ..... 65
Functional Requirements Identified from primary research. ..... 66

Functional Requirements Identified from secondary research. ....66
Non Functional Requirements. ....66
Framework Overview. ....67
Tuning Process. ....71
Construction Phase. ....73
Transition Phase ....78
Pretest Results. ....78
JMySQLTune Recommendations. ....79
Post-test Results. ....80
Conclusion ....81
Chapter 5- Discussion ....82
Introduction ....82
Summary of Questionnaire Research results ....82
Summary of the experimental pre-test post-test results ....83
Project Management ....84
Project Schedule ....84
Lessons Learnt ....85
Limitations ....86
Conclusion ....87
Chapter 6 - Conclusion ....88
Introduction ....88
Findings ....88
Project Recommendations ....92
Future Academic Research Recommendations ....93
Conclusions ....94
References ....96
APPENDICES ....110
Appendix A: Questionnaire ....110
Appendix B: SystemVariable.xml ....114
Appendix C Index.xml ....118
Appendix D: show status ....122

Appendix E: Explain command ..... 123
Appendix F: Explain select command ..... 124
Appendix G: MySQL Pretest Configuration ..... 125
Appendix H: Test Case Descriptions ..... 126
Appendix I: Hardware Configuration ..... 129
Appendix J: Pretest Results ..... 130
Appendix K: JMySQLTune Recommendations ..... 131
Appendix L: Post test Results ..... 133
Appendix M: JMySQLTune Main Flow Chart ..... 134
Appendix N: JMySQLTune Main Source Code ..... 135
ANNOTATED BIBLIOGRAPHY ..... 137

## List of Figures

Figure 1 JMeter MySQL Configuration ..... 52
Figure 2 Question 1 Result ..... 59
Figure 3 Question 2 Results ..... 60
Figure 4 Question 3 Results ..... 61
Figure 5 JMySQLTune Architecture ..... 68
Figure 6 JMySQLTune Sample decision tree ..... 70
Figure 7 JMySQLTune Iterative Tuning Process ..... 72
Figure 8 JMySQLTune Class Diagram (XML portion) ..... 74
Figure 9 JMySQLTune Class Diagram (Tuning Portion) ..... 75
Figure 10 Index.xml ..... 76
Figure 11 SystemVariable.xml ..... 77
Figure 12 Pre-test Response time in milliseconds of SQL Queries ..... 78
Figure 13 JMySQLTune Console ..... 79
Figure 14 Post-test Response time in milliseconds ..... 80

## List of Tables

Table 1 Participants Interest in an Index Tuning Advisor .....62
Table 2 Participants Interest in a Cache Tuning Advisor.....62
Table 3 Percentage of Participants Interested in using the proposed Product .....63
Table 4 Participants Experience of other Automated Database Tuning Tools .....64
Table 5 Current Method of Tuning MySQL Community Databases .....65

# Chapter 1- Introduction

## Statement of the Problem

This thesis will provide an automated database tuning tool, which will be used to solve the problems of identifying effective recommendations for index and system variable selection in MySQL community servers using scientific processes and repeatable techniques.

## Study Background

The use of databases has increased significantly over recent years and increasingly the information contained within these databases is being seen as containing mission critical to organizations (Chaudhuri & Narasayya, 2007).

One of the principle tasks of the database administrator is to ensure that database's under their supervision consistently deliver high performance. Manual tuning and optimization is a complicated task, requiring a deep understanding of the entire system. Without this knowledge, it is easy for the database administrator to cause severe degradation in database performance. The maintenance of database performance is further complicated as variances in workload and data distributions, often results in the existing configuration becoming suboptimal. Therefore, there is a requirement for database administrators, to periodically run a tuning tool capable of making recommendation to the current configuration (Bruno & Chaudhuri, 2006).

It has been the objective of many major DBMS vendors to offer tools which automatically and dynamically tune a database system. In SQL 7.0, Microsoft was the first of these to provide a physical design tuning tool, which they called an index tuning wizard. This was soon followed in 1999 by IBM DB2. By the time Oracle shipped its 10g release, a tuning

advisor which inputs a workload and then recommends indexes and materialized views based on the results, was included within its feature-set (Chaudhuri & Narasayya, 2007).

These tools enable the database administrator or indeed the application programmer, to "identify SQL performance issues and suggest actions to fix them" (Bayan & Cangussu 2004 p1099 p4). Techniques, adopted by these vendors, to achieve optimal performance, include simulation, black box, gradient descent and empirical equations (Tran, Huynh, Tay, & Tung, 2008). In August of 2007, MySQL AB began offering the MySQL server in two versions.

- MySQL Community
- MySQL Enterprise

Although, the code bases of the two servers are similar, there are also a number of significant differences. The most relevant of which for the purposes of this thesis, is the lack of support provided by Sun Microsystems for performance tuning of the MySQL Community server product. With MySQL Enterprise server, Sun Microsystems provide four levels of support from Basic to Platinum. Currently, only the most expensive Platinum version provides support for performance tuning (Sun Microsystems, n.d).

The MySQL Community server also differs from the Enterprise edition in that it offers freely downloadable binaries. The open source model, adopted by Sun Microsystems contains a development team, who release software to the open source community for the purposes of debug and test. Therefore, when MySQL Community databases experiences problems, the database administrator, often must rely on personal expertise and a growing user community to debug and determine a solution to the issue. This user community is a diverse global population of both users of and contributors to the MySQL community project.

In contrast to the MySQL Community server, many database vendors seek a competitive advantage, by providing additional features such as automatic performance tuning to as part of their database server solutions. This reduces both the total cost of ownership of the database and system downtime (Wiese, Rabinovitch, Reichert, & Arenswald 2008, Bruno & Chaudhuri, 2005 and Chung, & Hollingsworth 2004). One example of this is Oracle, who provides a wide range of performance tuning features such Automatic Database Diagnostic Management (ADDM), Advisors, Automatic Workload Repository (AWR), Automatic Segment Management and Shared Memory Management.

It is also true to say that MySQL Community Server suffers the fate of many of its open-source contemporaries. It offers advanced features, which in truth have not been implemented as robustly as the makers of high-end proprietary database (Paulson 2004). According to Annesley (2006) open source databases including MySQL, Cloudscape and Firebird are still years behind their commercial rivals in their ability offer a realistic, enterprise-ready alternative.

In 2008, Sun Microsystems was subject to a proposed takeover bid by Oracle. This in turn, has led to fears for the future of MySQL Community server. While Oracle is generally thought to support the open source movement, there is currently an uncertainty as to its future. MySQL Community server is likely to be reliant on manual intervention, for performance tuning for some time into the future.

This presents significant issues. From their qualitative research Chaudhuri & Narasayya, (2005) note that the most common cause of downtime in a database system is human error. The total cost of ownership (TCO) of information technology is increasingly dominated by people costs. Mistakes, in operations and administration of information systems, provide the single most common causes of system outage and unacceptable performance.

This thesis, will examine the MySQL Community Server tuning advisors to determine which configuration variables may be modified to increase performance. As in all databases, the major causes of performance bottlenecks are often due to poor database programming. This includes the use of irrelevant joins, performing unnecessary queries, returning unneeded data and performing unneeded calculations (Stephens & Russell, 2004). The isolation and removal of these poor programming bottlenecks are outside the scope of this thesis.

Within MySQL Community server, there is a number of Database Engine Tuning Advisor (DTA's). The types of information which may be extracted from these DTA's include recommendations for indexing, views, indexing on indexed views and horizontal range partitioning (Chaudhuri & Narasayya, 2007). The proposed tool, will therefore take advantage of these DTA's in order to identify and remove performance bottlenecks associated with missing or erroneous secondary indexing.

Secondary research will be reviewed and evaluated, to identify a practical approach to automated tuning, from this research, a hypothesis will be formulated. This will act as the basis for the proposed framework. The purpose of which, is to consistently yield better performing databases after being executed. Thus reducing the requirement for manual intervention when performance tuning MySQL Community Servers.

**Research Questions**

In this the thesis the following research questions will be answered

Q1. What problems are associated with manual intervention as a method for tuning MySQL Community database servers?

Q2. What database tuning methods currently exist which are carried out in a manual manner as part of a database administrators routine task on a MySQL database?

Q3. How can MySQL digital tuning advisors (DTA's) best be taken advantage of in order to identify possible tuning bottlenecks within a database?

Q4. Which of these tuning techniques are most suitable for automation thus reducing the total cost of ownership (TCO) of the MySQL database?

**Research Hypothesis**

H1. Automated tuning methodologies adopted by other vendors are applicable to MySQL Community Server.

H2. The resulting recommendations may be applied to a MySQL Community server.

**Importance of the topic**

In 2008 the Clermont report on database research stated, self-tuning is one of the most important areas of interest for future work in database research (Agrawal 2008).

This focus, on automatic database configuration, stems from organizations increasing reliance on the availability and reliability on the information contained within databases, whilst also reducing maintenance costs. This is further exasperated due to "the increasing complexity and scale of computing systems performance tuning that has been relying on expert's experience have been facing the limit. Therefore automated support for tuning is necessary" (Zenmyo, 2009 p53 p2).

At the same time, many organizations wish to use open source products such as MySQL Community server in order to further reduce costs. However, MySQL Community server lack of support for automated self-tuning, places the responsibility on human intervention to the database administrator. An over-reliance on manual intervention, presents significant challenges. According to Wiese et al (2008) mission-critical database systems must provide high performance consistently. This often results in organizations employing highly qualified

experienced and expensive database administrators, required to tune the system to the system workload characteristics.

The MySQL Community Server has a significant number of tuning parameters. For example the SHOW VARIABLES command on the MySQL 5.0.86-community-nt release contains a total of 235 configuration variables. To manually maintain a MySQL Community server at its optimal performance levels, would require a deep understanding of each of these configurations and the consequences associated with adjusting them. This not only overburdens the database administrator, but would also result in significant database downtime.

This reliance, on manual intervention comes at a significant cost. Lungu & Vatuiu (2005) believes human error is the most significant root cause of system downtime. The authors believe over 50% of unscheduled downtime can be directly attributed to human error. Therefore, an automated approach would significantly reduce such possibilities. This helps protect the system from the harm, such errors can cause. Lungu & Vatuiu (2005) believe this is the reason Oracle has invested significant resources in enhancing the manageability of their solutions.

The area of performance tuning, is seen by many, as one where the open source MySQL Community server fails to compete with licensed vendors such as Oracle or Sybase DBMS product solutions. This thesis, therefore will attempt to bridge this gap, by identifying a framework which will automatically tune parts of a MySQL database which currently require manual intervention.

The importance of such a product to MySQL users include

- It will reduce the possibility of human error
- It will reduce the amount of maintenance time required to perform tuning activities
- It will free up the database administrator expertise to concentrate on other tasks

- It will reduce the TCO of maintaining the performance of a MySQL database.

**Thesis Structure**

The thesis will be comprised of a total of six chapters. It will contain the following sequence

- Chapter 1:- Outlines the nature of the study.
- Chapter 2:- Presents reviews of the approaches discussed in academic literature with regards to tuning techniques and strategies for achieving high database performance.
- Chapter 3:- Discusses the methodology being applied in this thesis.
- Chapter 4:- Presents the results and analysis. This chapter commences with a technology evaluation review of other technologies. The results of the questionnaire will be then presented. This chapter also contains the level requirements, along with a presentation of proposed framework. Finally the chapter concludes with the results from the experimental pre-test post-test research.
- Chapter 5:- will discuss the results from chapter 4. Detailing research findings and presenting a discussion of the implications of these findings. This chapter will also present a set of recommendations for future development of the JMySQLTune product
- Chapter 6:- will conclude the study.

**Scope**

This thesis, through the literature review, examines several common aspects of database performance tuning. In particular, the factors commonly adjudged to cause poor database performance. In addition to this, problem diagnosis techniques will be evaluated and latest research in index tuning and indexing selection strategies will be reviewed. Finally, an analysis of the factors which determine system variable selection will occur.

This thesis, concentrates on Online Transaction Processing (OLTP) MySQL Community database servers. The researcher examines factors, which often results in poor performance in relational databases. From this analysis, the researcher presents a Java based framework with the purpose of automating the concepts that have been identified in the academic literature. The thesis attempts to reduce the requirement for manual intervention, in order to ensure that a MySQL Community database server is tuned to a level near the optimal.

During the secondary research, the researcher will focus on the key issues surrounding database performance tuning and how these issues can be identified and removed in an automated fashion. The researchers' primary objective during this research is to

- Use secondary research in order to clarify and define aspects that have important influence on improving performance in a relational database.
- Identify a Java based framework which will with the purpose of using the DTA's provided by MySQL and other methods identified during the research
- Identify indexes to be added, or re-write the query, or re-design
- Identify the slow queries
- Generate an execution plan which demonstrates the changes in performance

At the end of this thesis, the researcher should

- Demonstrate a clear improvement in the performance of the altered query/index
- Demonstrate that the alteration has in no way caused degradation in any other part of the MySQL database.

**Success Criteria**

The success criteria for this thesis will be measured, in a number of ways. Most importantly, the thesis should provide a framework capable of providing recommendations for

index and system variable selection, in MySQL community servers. A "tuned" database will then be assessed, to determine if the recommendations results provide actual performance improvement. Finally, during the stability phase tables/queries unaffected by the tuning process will be evaluated in order to determine if changes have resulted in degradation.

The next chapter presents a review of relevant contemporary literature, focusing on different methods under research in the area of database performance tuning.

# Chapter 2 – Literature Review

## Introduction

This chapter will commence, by examining the most common factors which have been noted to result in performance bottlenecks in database servers. Once this is complete, the researcher examines literature, in the area of problem diagnosis techniques. The researcher will then evaluate a number of tuning methodologies, which have been subject to recent interest. Off-line and online tuning will be discussed, the issues surrounding index selection and memory configuration will also be examined. This chapter will conclude, discussing the utilities which are available on a MySQL community server which may be useful for performance tuning.

As database system usage has increased in recent years so too has the complexity of the database administrator's task. Physical tuning has become more complex and so database administrators increasingly struggle to identify and implement suboptimal solutions (Bruno & Chaudhuri, 2005). During the time same period these database systems have become mission critical to many organizations (Weikum, Moenkeberg, Hasse, & Zabback, 2002).

The term physical database tuning can be defined as altering the physical database definition and it typically includes construction, dropping or renaming of a non unique indexes or conversion between indexing and hashing (Sockut & Iyer, 2009).

The area of automated physical tuning of databases has a long history and indeed been subject to academic research since the mid 1970's (Hammer & Chan, 1976). In the subsequent years, researchers have identified many different tuning methodologies with varying degrees of

success (Hammer & Chan, 1976, Elnaffar, 2002, Bruno & Chaudhuri, 2005, Schnaitter, Abiteboul, Milo, & Polyzotis 2006, Bayan 2008, Wiese et al., 2008).

Although academic research into this field has continued over a long period of time new methods and ideas continue to be offered as possible tuning solutions. Even within the last decade, new areas of research has identified the use of materialized views, partitioning, merging, enumeration, tuning alerter, workload sequencing and dynamic online tuning have received particular attention (Chaudhuri & Narasayya, 2007).

This research, has led many database vendors to include automated tuning solutions as part of their products feature sets. The aim of these products is to reduce the DBMS's total cost of ownership, moving the burden of physical tuning from the database administrator to the database server (Bruno & Chaudhuri, 2005). MySQL Community server is an exception to this rule.

Unlike its competitors, MySQL Community server is an open source database server product, freely downloadable from the Sun Microsystems website. According to Annesley (2006) the open source database model experience differs fundamentally from that of the operating system. The author states, while open source operating systems are often used to run commercial databases, the development of open source databases is not advanced when compared with their commercial rivals. One area where MySQL Community lags behind is performance tuning.

Although free to download, research indicates, MySQL Community servers lack of support for automated physical tuning will contribute to both increased total cost of ownership and system downtime (Bruno & Chaudhuri, 2005, Weikum et al., 2002). In today's E-Commerce environment, poor performance often results in losing potential and existing

customers, as these people are becoming less likely to accept slow server responses (Weikum et al., 2002). In order to counteract this lack of support for automated tuning, the database administrator must constantly monitor and assess the database. This work is carried out in the realisation that over the lifecycle of typical database server, reorganization should occur several times (Sockut & Iyer 2009, Bruno & Chaudhuri, 2005).

This is an unsatisfactory approach, as mission critical systems within organizations are being subjected to the 'black art' or adjusting knobs and settings relying on the database administrator's ability to determine and implement an acceptable configuration (Weikum et al., 2005).

There are four areas of a MySQL Community server where the database administrator may tune to increase performance. These areas are SQL tuning, schema and index tuning, server tuning and operating system tuning (Cabral & Murphy, 2009).

**Factors which cause poor performance in relational databases**

Academic research has resulted in a rich set of literature which attempts to automate the identification of the key factors which result in poor database performance (Wiese et al., 2008, Bruno & Chaudhuri, 2005 and Chung & Hollingsworth, 2004).

Researchers agree, before physical tuning commences, it is essential to identify the current bottlenecks contained within the database server (Bayan & Cangussu, 2005, Zawodny & Balling, 2004). Therefore problem diagnosis should be considered as a crucial element of the tuning process and as such, it is the starting point for any automated tuning solution (Bayan & Cangussu, 2005).

Performance problems, which most commonly result in bottlenecks in a database server, are typically as a result of lack of database administrator expertise (Bruno & Chaudhuri, 2005,

Zawodny & Balling, 2004 and Weikum et al., 2002). There is a failure to find an optimal configuration. This is largely due to inexperienced database administrators. These individuals often fail to fully understand all of what Zawodny & Balling (2004) believe to be the most significant causes of poor database performance. These performance bottlenecks are disk I/O, CPU usage, memory latency and incorrect indexing.

Disk I/O is typically the most common cause of bottlenecks in database server performance. The root cause of these being, inefficient queries where the DBMS has to read too many rows to locate the select data (Zawodny & Balling, 2004).

Much research has occurred in order to provide solutions for the removal of these bottlenecks (Manegold, Boncz, & Kersten 2000, Weikum et al., 2002 and Sockut & Iyer, 2009). The memory latency and incorrect indexing will be discussed in more detail later on during this literature review.

**Problem Diagnosis**

In order to tune the MySQL database we first must be able to determine and diagnose the problems which exist within (Wiese et al., 2008, Bruno & Chaudhuri, 2005 and Chung & Hollingsworth, 2004). Wiese el al., (2008) emphasise this point, declaring, accurate problem diagnosis is crucial in any best practice autonomic tuning system.

Despite the extensive research which has already occurred in the context of automatic database tuning, Qiao, Soetarman, Fuh, Pannu, Cui, Beavin (2007) conclude, a shift in emphasis is required by many database vendors. They argue, this is required, to achieve a truly autonomic problem detection solution. Too many database systems provide raw performance data and therefore rely too heavily on the capabilities of the database administrator to analyse the problems which exist within.

It is however, not entirely true to say, no research has occurred in the area of autonomic problem diagnosis. Bayan & Cangussu (2005) proposed a data mining type approach, specifically the use of diagnosis rules in order to diagnose performance problems, identified whilst measuring workload. According to the authors adopting a rule based approach, allows resources to be considered as candidates for diagnosis. Diagnosis trees are utilized, in order to analyze the information gathered during the workload phase. This information, combined with rules, is then used to determine the source of the performance problem (Bayan & Cangussu, 2005). This approach, complies with by Zawodny & Balling (2004) belief that diagnosis of poor performance can only commence when it is determined how each resource (Memory, CPU, disk I/O) performs with a particular workload.

According to Qiao et al (2007) policies may be used as 'directives' to the underlying database. These policies direct the database actions under pre-determined circumstance. The author suggests such a framework, would provide benefit in problem detection, as they could be used to identify events, such as exceptions and therefore aid in early problem diagnosis.

Elnaffar (2002) provides us with a practical demonstration of the decision tree approach. This provides an insight, into how such a solution may be adopted in the DBMS environment. In this case, Elnaffar demonstrates the value of decision trees, using them to distinguish between Decision Support System (DSS) and Online Transaction Processing (OLTP) workloads, in a relational database. According to the authors the building of such a tree consists of two individual phases. Initially a model is built by analyzing a set of data objects. This model is then transformed in to a decision tree representation containing the rules used to categorize future data objects. Elnaffar (2002) also suggests the use of 'pruning' in order to remove any spurious branches.

**Tuning methodologies**

The physical tuning problem is defined by Bruno & Chaudhuri (2006) as being presented a representative workload of queries and updates and a space constraint. The task of the database administrator is to seek a physical configuration which fits in within the space and provide the lowest estimated possible query execution cost.

Like Chaudhuri & Narasayya (2007a) this thesis made the assumption that the workload is a set of queries and updates. Whilst the problem itself can be easily defined, the solution is less transparent. According to Bayan & Cangussu (2005) the DBMS tuning tasks consists largely of two phases, diagnosis and resource adjustment. The determination of any tuning plan, depends on the information extracted during these phases and therefore essential to implement a best-practice tuning system (Wiese et al., 2008).

Before automated tuning became of interest to academics, database administrators commonly used the trial and error approach. Even today, Bruno & Chaudhuri (2005) notes that trial and error as one of the most common and manual tasks employed to tune a database. However, this approach has a number of significant drawbacks.

Firstly, it relies on the database administrators abilities, to measure performance, change configuration and finally re-measure. In effect, without the database administrator, having a high understanding of tuning techniques, the database, on which he/she works, would make this approach, impractical. Bruno & Chaudhuri (2005) note that although this method will eventually result in the optimal configuration, the sheer volume of parameters to tune is so large that it would take too long to complete in practice.

Support for Bruno & Chaudhuri (2005) point of view is provided by Chung & Hollingsworth (2004). The authors point out that when using a trial and error approach a system containing a total of 10 parameters, with each parameter having 2 possible values, would result in a search size space of $2^{10}$.

The majority of recent research on database performance tuning, has tended to concentrate, in the realm of self regulation. He, Lee, & Snapp (2005) state, that in the context of the database server this can be taught of, as to refer, to databases ability, to automatically tune based on both application and hardware capabilities. This in turn would result in a significantly reduced administration overhead. Many of the latest approaches have significant similarities, in that they attempt to concentrate on estimating and incrementally updating the tuning parameters based on query feedback.

The use of heuristics is one of the most important tools identified by academic research in the areas of performance tuning. Many of the major licensed vendor's solutions, including Oracle, rely on heuristics to achieve an optimal configuration (Chung & Hollingsworth 2004, Bruno & Chaudhuri, 2005). One of the principal advantages, of this approach is heuristics has the ability to reduce the search space. This allows tuning operations to be completed more efficiently (Slivinskas, Jensen, & Snodgrass, 2001). Vuduc, Demmel, & Bilmes (2004) reminds us, that heuristic performance modelling is not necessarily something which exists in isolation and therefore in practice it is often used in combination with other evaluation techniques.

This has enabled academic research to continue evaluating other possible techniques. This in turn, has led to the identification of a number of different methodologies, which may be used in isolation or in conjunction with heuristics, in order to formulate a tuning plan for relational databases. Examples of such possible solutions include mathematical techniques,

relaxation based approach or even a guessing based approach (Wiese et al., 2008, Chaudhuri & Narasayya, 2007, Osogami & Kato, 2007, Chung & Hollingsworth, 2004, Bruno & Chaudhuri, 2005, Weikum et al., 2002).

Wiese et al (2008) suggests, any automated self regulation framework should include support for "implemented tuning plans cover the areas of online resource allocation and configuration (heaps, caches, locklist), physical database design, as well as statement tuning" (Wiese et al., 2008 p9 p6).

Weikum et al (2002) favour a mathematical approach, for DBMS performance tuning. The author's suggestion is that, performance tuning quantitative tendencies makes it a candidate for mathematical modelling techniques. The author's present a persuasive argument, that any tool should also contain a feedback control loop, which constantly monitors certain predefined parameters. While recognizing the value of heuristics and even the addition of additional hardware resources the authors conclude mathematical techniques minimize the possibility of overreacting to performance fluctuations.

According to Bruno & Chaudhuri (2005) a relaxation based approach, begins, by configuring, an optimal configuration which is larger than the space available. Then, using an iterative methodology transforms the configuration so that it consumes fewer resources, but is also less efficient.

In contrast to this, Osogami, & Katosuggest (2007) simply guessing the optimal configuration, by use of an algorithm, are coining the term Quick Optimization via Guessing (QOG). This approach is based on the theory that a somewhat optimal performance, may be arrived at by guessing, based on the results of similar configurations. Once the suboptimal configuration is found, the measurement of a configuration is terminated.

Chung & Hollingsworth (2004) believed that the utilization of historical data may also be of benefit. This view is supported by Chaudhuri & Narasayya (2007), who identified work load analysis as one of the more significant developments in database tuning technologies over the last ten years.

With such a wide range of complex competing theories, it becomes obvious, that there is currently more than one method which may be adhered to, in order to achieve optimal database tuning performance. Although each method has its own advocates, such a wide range of competing approaches, has in reality furnished confusion (Wiese et al., 2008).

According to Wiese et al (2008) database server physical tuning would benefit, from adopting a method similar to the Standard Operating Procedures (SOP's) from the health industry. The authors believe, the use of such agreed best practices by the database community would provide, significant progress in the areas of autonomic DBMS tuning.

**Online versus Off-Line Tuning**

Traditionally, physical database tuning tended to occur during regular maintenance intervals and therefore most legacy techniques, favour the off line approach (Zenmyo, 2009, Schnaitter et al., 2009). According to Sockut & Iyer (2009) the choice whether to adopt the online or off line approach depends solely on the availability requirements of the database. The authors summarize, it be unacceptable to take a highly available system down for regular maintenance activities.

As external business requirements grows for mission critical databases, to offer high availability, so too has the interest in the On-Line tuning approach (Zenmyo, 2009, Schnaitter et al., 2009, Sockut & Iyer, 2009 and Chaudhuri & Narasayya, 2007). In fact, Chaudhuri & Narasayya (2007) identify dynamic online tuning, as one of the most important advances, in

physical database tuning over the last 10 years. This tuning methodology, adopts an always on approach, which requires little or no manual input and therefore, reduces the possibility of human error.

However academic research into the online tuning approach, determines that this is only a feasible solution, when the database server itself supports self regulation of physical tuning (Schnaitter et al., 2006, Sockut & Iyer, 2009).

**OLTP versus DSS**

Before tuning commences, one should consider the whether the database is Online Transaction Processing (OLTP) or Decision Support System (DSS) (Elnaffar, 2002, Storm, Garcia-Arellano, Lightstone, Diao, & Surendra 2006).

The difference between the tuning required for Online Transaction Processing (OLTP) and Decision Support Systems (DSS) is significant. The tuning of system resources is altered significantly based on this characteristic alone (Elnaffar, 2002, Storm, Garcia-Arellano, Lightstone, Diao, & Surendra, 2006).

According to Elnaffar (2002) one can distinguish workload, based on the hypothesis that certain attributes may be used to determine workload characteristics including log size, locking and index usage.

**Selection of Digital Tuning Advisors**

According to Chaudhuri & Narasayya (2007), the selection of the correct statistical information, as being the crucial element in the determination, of a suitable indexing plan. However the authors also acknowledge the complexity, associated with this task. This leads the authors to conclude, the quality of decision making in the selection of digital tuning advisors, has

a direct relationship to the quality of the final physical performance tuning solution. MySQL provides a number of such tuning advisors (Zenmyo, 2009).

**The addition of Physical Hardware**

One of the most common solutions adopted in the real world environment is configuring additional physical hardware resources (Manegold et al., 2000, Weikum et al., 2002). Indeed academic research agrees bottlenecks such as memory latency may be reduced by the configuration of additional cache memories in the memory system (Manegold et al., 2000).

This has led many database administrators, to view the addition of extra computer hardware such as memory and/or processing power, as a feasible solution to performance bottlenecks. This has been facilitated by computer hardware, continued price reduction in and growth in size. This is turn enables the database administrator to apply a 'kill it with iron' or KIWI type approach which ultimately may avoid problems such memory latency (Weikum et al., 2002).

Manegold et al (2000) supports this view, demonstrating using a load testing technique, the effect of memory latency in relation to presenting bottlenecks for database performance. Using these results as a baseline, the authors further demonstrates how the addition of cache memory can be used to demonstrate better database performance (Manegold et al., 2000).

However despite this success, the "kill it with iron" approach does not utilize the hardware systems resources effectively or efficiently. This is in fact a solution, which tends to hide the underlying programming and configuration problems associated with the database (Weikum et al., 2002). Therefore a more technically desirable approach, to improving database performance, is through efficient index creation (Sockut & Iyer, 2009, Di Giacomo, 2005).

**Indexing Selection Strategies**

According to Sockut & Iyer, (2009) the reorganization and construction of indexes, is one of the principal maintenance tasks assigned to the database administration team. The significance of indexing cannot be understated. When used appropriately, indexes have the ability, to match rows efficiently. MySQL stores all indexes in B-trees and as such, can index all column types (Di Giacomo, 2005).

According to Bujdei, Moraru, & Dan., (2008), one of the most important tools at the disposal of the database in increasing database performance is the correct creation and usage of efficient indexes. However, academic research also acknowledges correct index selection is not a trivial task (Sockut & Iyer, 2009, Di Giacomo, 2005, Hammer & Chan, 1976).

The challenges associated with index reorganization have been researched as far back as the 1970's. In this decade, Hammer & Chan (1978) discussed the associated complexities, noting, when used appropriately secondary indexes can improve the execution time of the queries accessing it. At the same time, Hammer & Chan (1978) conclude, when used inappropriately, the same index will ultimately cause degradation of the entire database server performance. According to Comer (1978) any solution to the index selection problem must also consider factors such as file organization, the number and type of transactions conducted, as well as the cost of index creation and maintenance along with the potential of index to decreasing access costs.

Di Giacomo (2005) warns of other potential pitfalls, such as indexing with CHAR and VARCHAR type fields in MySQL. The author points out that it is usually a less time consuming and less costly approach, to index a column prefix rather than the entire column.

Despite this complexity, index selection, remains the most important factor in speeding up database performance (Di Giacomo, 2005, Talebi, Chirkova, & Stallmann, 2008 and Kormilitsin et al., 2008). Indeed, most modern automatic physical database performance tools concentrate on providing recommendations on both indexing and materialized views (Bruno & Chaudhuri, 2006). An index can be a single or many columns of a table, depending on structure and type of queries which are being executed frequently on that table (Brunei et al., 2008).

Research suggests, evaluating and implementing these recommendations on the physical database, is an essential part of the database administrator's task. Indexes must evolve to suit the databases table sizes and workloads. This results in indexing reconfiguration occurring many times over the lifetime of a database (Chaudhuri & Narasayya, 1997, Finkelstein, Schkolnick, & Tiberio, 1998). According to Zaman, Surabattula, & Grunewald (2004) any small change in workload pattern may require a re-indexing of the database as the current indexing become sub-optimal.

It is essential that any indexing tool provides a selection strategy, which when implemented does not degrade the current performance levels (Di Giacomo, 2005, Bruno & Chaudhuri, 2006 and Talebi et al., 2008). However, the issues surrounding automatically or indeed manually choosing such an indexing strategy is a complicated task (Chaudhuri & Narasayya, 1998, Di Giacomo, 2005, Bruno & Chaudhuri, 2006, Talebi et al., 2008). They must take into consideration the interactions among other indexes and the amount of storage space used by indexes means the choosing of irrelevant indexes will in fact result in performance degradation (Ip, Saxton, & Raghavan, 1983).

Some early research, into indexing strategies, favoured mathematical approaches (Lum, 1974). The authors base their hypothesis behind a mathematical formula which used the

simplistic yet effective theory that index selection should be made based on two factors, that is gain in terms of retrieval time when selecting an attribute from a secondary index versus the cost of inserting deleting and updating attributes contained within that index.

After over twenty years of further academic research, Chaudhuri & Narasayya (1997) note that early mathematical theories such as Lum (1974), may have been somewhat naive in their analysis. According to Chaudhuri & Narasayya (1997) rich evidence exists, to suggest, such mathematical approaches, failed to consider a critical element in their calculation. This critical element is simply workload (Comer et al., 2008, Chaudhuri & Narasayya, 1998). Researchers agree, this failure meant that the second class of index selection tools ultimately failed to achieve their goals due to the failure to take account of the information which may be extracted from the query optimizer (Chaudhuri & Narasayya, 1998, Finkelstein el al., 1998).

Much of the recent academic research, has revolved around the selection of indexes and materialized views in order to enhance performance (Chaudhuri & Narasayya, 1998, Finkelstein el al, 1998, Agrawal et al., 2001, Bruno & Chaudhuri, 2006, Kormilitsin et al., 2008).

According to Chaudhuri & Narasayya (1998), one possible approach entailed removing spurious indexing at an early stage. In contrast to other index selection tools alternative complex indexes are evaluated in an iterative manner from simpler alternatives. The author's states that any column used in a query are indexable columns.

In contrast, Agrawal et al (2001) implemented an architecture which uses indexes, materialized views and indexed material in order to optimize the index selection policy. This approach operates by selecting likely indexing candidates. Then through analysis of each candidate, produce a recommendation which considered the benefits of indexes and materialized views for a given workload.

In their research finding, Bruno & Chaudhuri (2006) suggest a method for determining the indexing strategy, by intercepting a databases index requests. From these requests the authors suggest storing four separate variables for each index

- the set of columns in the sargable predicates (S)
- the sequence of columns for which an order has been requested (O)
- the additional columns used in the execution plan (A )
- And finally the number of times the sub plan is executed (N).

From the combination of these four stored variables Bruno & Chaudhuri (2006) coin the term SOAN. The critical thinking behind the authors' hypothesis is that SOAN can be used to calculate the cost of an alternative sub-plan.

Despite this wealth of research, academics believe, the quest for and the optimal configuration may be just an aspiration (Talebi et al., 2008). According to some researchers "the rule of thumb" or heuristics will for the foreseeable future remain a preferable approach (Weikum et al., 2002, Osogami & Kato, 2007).

Although the use of greedy and other heuristic strategies for index and view selection may in a lot of circumstance have been deemed to be the most feasible solution (Weikum et al., 2002, Talebi et al., 2008 and Bujdei et al., 2008) choosing the correct heuristics is still deemed to be a difficult task (Chaudhuri & Narasayya, 2007).

In their research Bujdei et al (2008) suggests, the following heuristics should be imprinted in the database administrator's consciousness when determining the indexing strategy.

- The column which has a short range of values should be inserted first in the index
- A table should generally have no more than 5 indexes,
- It is better to use in an index columns with type integer value

- The time of response is inverse proportional to the index size
- Do not use indexes on tables which contains a small number of records
- Indexes slow down the insert, update and delete operations.
- Where possible use a clustered rather than a non-clustered index, if the results are sorted using GROUP BY or ORDER BY syntax (Bujdei et al., 2008)

Despite the success of materialized views in increasing database performance MySQL community version as of yet does not support such a feature. Therefore, it is not deemed to be relevant for consideration in the JMySQLTune product.

**Memory Allocation**

Memory latency is one of the most significant causes of bottlenecks in many live commercial database systems (Zawodny & Balling, 2004). This is often due to systems which have their database memory configured to a somewhat suboptimal level (Dageville & Zait, 2002, Storm et al., 2006 and Tran et al., 2008).

Database memory is used in database system to increase system performance by a combination of reducing latency, decreasing I/O or decreasing contention (Storm et al., 2006). Optimal memory tuning benefits database servers as complex database queries rely heavily on memory intensive operators (Dageville & Zait, 2002, Storm et al., 2006 and Tran et al., 2008). Currently many commercial database systems use manual database administrator's expertise for the configuration of memory (Dageville & Zait, 2002).

Therefore the task of allocating memory to database servers has traditionally been assigned to the database administration team. This has led to many database systems containing configurations which are tuned to a suboptimal level (Storm et al., 2006, Tran et al., 2008). Academic research has concluded this is largely due to two separate issues. Firstly, variances in

customer workloads are unpredictable. This in turn produces an unpredictable demand on memory resources. Secondly, the tuning process consists of database administrators, altering parameters and running workloads (Storm et al., 2006, Tran et al., 2008).

During each workload run, results are analyzed by the database administrator, to determine if the new configuration is at the optimal level. This is a time consuming process, which relies too heavily on the capabilities of the database administrator (Storm et al., 2006).

Although much academic research has occurred around the area of memory tuning for databases, this research, has resulted in approaches have resulted in solutions not deemed practical for commercial database systems (Storm et al., 2006, Tran et al., 2008). One common area of research is the use of heuristics for the purpose of choosing cache sizes, based on workload. However, further research indicates this methodology does not produce reliable results (Weikum et al., 2002, Storm et al., 2006). Once the workload changes, the heuristic configuration becomes suboptimal. Another criticism levelled towards heuristics is that, the approach adopted by academic researchers is too narrow and therefore impractical in real world scenarios (Storm et al., 2006).

Despite the complexity of the problem, many commercial database vendors have attempted to include automated memory tuning in their product feature sets (Storm et al., 2006, Tran et al., 2008). These vendors typically use one of the following memory tuning techniques, simulation, black-box control, gradient descent or empirical equations (Tran et al., 2008)

Trace simulation is often used in commercial database systems to estimate I/O costs for different buffer allocations (Storm et al. 2006). Tran et al (2008) critique the trace method, noting that in some cases traces are unavailable. In addition to this simulation code is hard to modify (Tran et al., 2008).

The black-box control method is based on an iterative loop which treats the buffer as an unknown function from buffer allocation to *P*miss (Ko, Lee, Amiri, & Calo, 2003). Tran et al (2008) contends that convergence cannot be guaranteed, or in some cases be slower than the dynamic workload changes.

The gradient descent is an iterative technique which uses a measured gradient in memory (Ko et al., 2003). One criticism of this technique is that in certain circumstance fluctuations occur which have been noted to result in large gradient errors, which in turn destabilizes the convergence (Tran et al., 2008).

While empirical equations don't have a theoretical basis, they are chosen for their ability to fit data. One example of empirical equations is Belady's power law (Tran et al., 2008, Storm et al., 2006). Critiques of empirical equations note, that they provide a poor fit for database workloads, in addition there is an upper buffer space limit needed by any workload which these equations fail to consider (Tran et al., 2008).

Weikum et al (2002) mentions the use of a five minute rule, to determine a minimum cache level based on workload and cost. However, the authors point out most commercial systems often use significantly larger cache sizes, than the results produced by this rule, in order to increase throughput.

Recent research concludes, to determine the memory allocation, one should consider three separate variables, workload, buffer size and buffer share (Storm et al., 2006, Tran et al., 2008). These variables are then evaluated using mathematical techniques, in order to resolve the key issues associated with the allocation of memory to the buffer for caching data and disk page (Tran et al., 2008).

Storm et al (2006) favours the trace simulation method. In this approach the total amount of memory available to the database is altered in order to find a near optimal memory distribution. Trace simulation, is then used to estimate I/O costs for different buffer allocations. The reasoning behind Storm et al (2006) hypothesis is, there should be a concentration on the distribution of memory for operations such as sort, hash join, SQL cache, lock memory, buffer pools, compilation memory and statistics memory. The criteria for choosing these configurations are solely based on workload characteristics observed at run time.

In their research Tran et al (2008) suggest an approach to which is based on a buffer miss equation. In this case the hypothesis is to fit the available data within the equation. This equation is then derived from an analytical model and unlike empirical equations the authors identify the upper bounds on buffer size which are considered applicable for tuning purposes.

**MySQL Database Monitoring and Tuning**

According to Cabral & Murphy (2009) tuning of the MySQL database server is done via the configuration file, which contains both static and dynamic variables. The static variables may be modified by editing the configuration file manually, or by starting the database server with certain options (Kofler & Kramer, 2005). Dynamic variables are modifiable both in the configuration file and by using SQL commands (Cabral & Murphy, 2009).

The SHOW VARIABLES command is often used to retrieve the current values of MySQL system variables. Although this command contains a wealth of information with regards the status of the database server, only a subset of is relevant for performance tuning. To aid diagnostics of how well these variables may be tuned, MySQL also provides the SHOW STATUS command (Cabral & Murphy, 2009).

The SHOW STATUS command is used to retrieve the value of over 250 status variables for a particular session. While adding the GLOBAL modifier SHOW STATUS shows the status for all connections to the server (Stephens et al., 2004).This command is valuable as it can be used to determine if the MySQL Community server is demonstrating poor performance (Zawodny & Balling, 2004).

Each of these system variables has a predefined default value. It is possible to configure system variables at server start-up using command line options or in an option file (Kofler & Kramer, 2005). However a large proportion of these variables may be altered dynamically by means of the SET statement. This enables the database administrator to alter the database server's operation without having restart (Cabral & Murphy, 2009).

**MySQL Cache Memory Tuning**

There are many obstacles towards achieving an optimal memory configuration in any relational database (Dageville & Zait, 2002, Storm et al., 2006 and Tran et al., 2008). According to Kofler & Kramer (2005), MySQL Community server provides a number of variables in the SHOW STATUS command which may be used to establish the health of the current configuration.

At start-up MySQL reserves a portion of main memory for certain tasks. The size of this buffer is controlled by options in the configuration file. Poor configuration of the system variables results in MySQL leaving a high proportion of this memory unused (Kofler & Kramer, 2005).

A cache table is a regular table in a database that stores the results of one or more queries for faster retrieval. Common query results, which could be cached in a table, include, counts, ratings and summaries. One common performance metric used to monitor memory performance

is the cache hit ratio. This metric provides information as to the ratio of page accesses which require disk reads (Tran et al., 2008).

MySQL database server uses a cache mechanism to store the most frequently accessed table blocks in main memory (Cabral & Murphy, 2009). The purpose of this cache is to minimize disk I/O issues, one of the major causes of poor performance in any database server (Zawodny & Balling, 2004). MySQL contains a number of cache tuning variables which may be altered using dynamic or static configuration methods (Kofler & Kramer, 2005, Cabral & Murphy, 2009).

According to Cabral & Murphy (2009) MySQL cache memory can be tuned at a number of different levels including

- key_buffer_size
- table_cache
- query_cache_size
- thread_cache_size
- sort_buffer
- tmp_table_size

A special cache structure, called the key cache is maintained for index blocks. This structure consists of block buffers, where the most-used index blocks are placed. If the key cache is not configured, or not working correctly, the index blocks are accessed using the file system buffering mechanism provided by the operating system. The key_read_requests status variable contains the number of physical reads of a key block from disk. In order to determine if the key_buffer_size cache value is too small, the cache miss rate is calculated. This is done using the formulae variable Key_reads/Key_read_requests (Kofler & Kramer, 2005, Cabral & Murphy, 2009).

According to Kofler & Kramer (2005), key cache is tuned to the optimal value, when the value of the Key_reads divided by the value of key_read_requests produces a result less than 0.01. Also, dividing the value of key write by the value of key_writes_requests should produce a result less than 1 (Kofler & Kramer, 2005).

Each time MySQL accesses a table, it places it in table cache. If the system accesses many tables, it is faster to have these in the cache. MySQL, being multi-threaded, may be running many queries on the table at one time and each of these will open a table. According to Cabral & Murphy (2009) to determine if the table cache is correctly sized, the Open_tables and Opened_tables status variable is checked. The Open_tables variable contains the number of tables which are currently open. The Opened_tables variable contains the number of tables which have been opened since the server was last restarted. These two variables can be used to determine if the table cache is configured to the optimum level. A high value for Opened_tables indicates the table_cache variable is too low and requires manual intervention (Kofler & Kramer, 2005, Cabral & Murphy, 2009).

In some situations the database repeatedly runs the same queries on the same data set, which in turn provides the same results each time, MySQL can cache the result set, avoiding the overhead of running through the data over and over and is extremely helpful on busy servers. The purpose of this qcache_hits variable, is to record the number of accesses to the new MySQL query cache. The query cache retains the results of frequently-used queries in order to speed up database response time (Kofler & Kramer, 2005, Cabral & Murphy, 2009).

The MySQL database server maintains a cache of unused connection threads in the thread_cache_size variable. This configuration is relevant in database servers with quick

connections. By setting the thread cache at an appropriate value high enough that the Threads_created value in SHOW STATUS will not become excessive (Cabral & Murphy, 2009).

The sort_buffer determines the amount of memory which will be allocated to the database server for SQL sort operations. This is a useful everyday operation where the database server is performing large numbers of sorts (Cabral & Murphy, 2009).

The tmp_table_size cache combined with the max_heap_table_size setting is used to determine the maximum size of a memory temporary table before it is converted to a MyISAM table. The smallest value for these two settings is the one utilized (Kofler & Kramer, 2005, Cabral & Murphy, 2009).

Although other options, such as altering memlock, have the potential to increase performance, it also potentially creates instability. It achieves this by locking mysqld into system memory and therefore it cannot be stored into swap space (Cabral & Murphy, 2009).

**MySQL Tuning**

Generating reliable index recommendations for a MySQL database in an automated manner is one of the principle objectives of this thesis. As we have seen during the literature review there are a number of different theories on index selection (Chaudhuri & Narasayya, 1997, Finkelstein et al., 1998, Agrawal et al., 2001, Bruno & Chaudhuri, 2006 and Kormilitsin et al., 2008). However, all these selection policies will benefit from the information extracted from the MySQL database.

The status of indexing is evaluated using variables such as show_queries and select_scan. The show_queries variable contains the number of queries which have taken longer than the predefined per_query time limit to execute. A large value here indicates that the server is not able to process queries at the speed it should and is a cause for concern. This determination is

only made, after checking the slow query log which contains the actual query strings, to see if it's the queries themselves that are unusually long or complex (Cabral & Murphy, 2009).

The slow query log file contains the precise SQL Statements such as SELECT, INSERT, UPDATE, DELETE and JOIN which take longer to run than a timer called the long_query_time configured in the system. During high workloads it is probable that examining the long slow query log would become a difficult task. In order to counteract this, MySQL community server also provides a utility named mysqldumpslow. This utility can be then used to parse the MySQL slow query log files and prints a summary of their contents (Cabral & Murphy, 2009).

Other information which can be extracted from these tables for index tuning includes select_scan and select_full_join. The select scan variable contains the number of join operations and if a full table scan was required. Full table scans are resource heavy and time intensive operations. Therefore a high value in this statistic indicates queries are operating in an inefficient manner and therefore are possible candidates for optimization (Cabral & Murphy, 2009).

The select_full_join statistic contains the total number of joins which were performed not making use of indexes. The purpose of indexing is to speed up table searches and therefore it is usually advisable to ensure indexes are created on all fields which are queried frequently. A high value in this variable indicates the database is not using indexes correctly and therefore is a possible candidate for optimization. This problem is usually resolved by evaluating and indexing important fields of a join operation (Cabral & Murphy, 2009).

Another command which is useful for index selection in MySQL databases is ‘SHOW TABLE STATUS’. Like the ‘SHOW TABLE’ command, ‘SHOW TABLE STATUS’, provides detailed information with regards to non-temporary tables contained within the database. The

'SHOW TABLE STATUS' command returns such information as, the name of the table, the storage engine, the row format, the number of rows, the average row length, the data file length, the maximum data file length, the index file length, and the number of unallocated bytes (Cabral & Murphy, 2009).

EXPLAIN is a query analysis tool, provided by an SQL extension. This analyzes a query and provides information such as the number of many tables involved, how many joins occurred, how data is retrieved, sub query information, union information, clauses, temporary table information and indexing. In addition, the number of records and sorting information are also provided. This information is known as the query execution plan or EXPLAIN plan (Cabral & Murphy, 2009).

This EXPLAIN utility may be also used in two separate contexts. In the first context the EXPLAIN key word can be used to describe the contents of a table. Appendix E demonstrates a sample of the type of information extracted from the MySQL Community server using the 'Explain Table' command. It is invalid to use EXPLAIN in front of data manipulation statements such as UPDATE, INSERT, REPLACE or DELETE statements. Therefore in order to test DML statements they must be transformed into a corresponding SELECT statement (Cabral & Murphy, 2009). Appendix F demonstrates a sample of the information extracted from the MySQL Community server using the 'Explain Select' command.

**Chapter Summary**

During the literature review, the researcher examined a number of issues relevant to the physical tuning of MySQL community servers. According to Bruno & Chaudhuri (2005) database administrators very often suffer from a lack of understanding in relation to these systems and as a result make suboptimal design choices which can lead to major performance

issues. The researcher found, although automated tuning of databases has received much attention in the recent past, no silver bullet currently exists to solve this problem.

Much of the latest academic research in this field has concentrated on the area of online dynamic tuning approach (Zenmyo, 2009, Schnaitter et al., 2009, Sockut & Iyer, 2009 and Chaudhuri & Narasayya, 2007). However this approach is only relevant when the database is self regularity.

According to Bruno & Chaudhuri (2006) the search for an optimal configuration is impractical. Therefore the use of heuristics to find a near optimal configuration, presents a more realistic approach to solving the tuning problem. Elnaffar (2002) suggests using a decision tree approach provides an understandable methodology to solve complex questions. Finally Cabral & Murphy (2009) provides some details on how to best extract performance from the MySQL Community server.

The next chapter contains details of the research and development methods used during the course of this project.

# Chapter 3 – Methodology

## Introduction

The thesis is attempting to research and design a framework capable of providing recommendations for the physical tuning of MySQL community servers. Typically in design research there are a number of distinct tasks executed to reach the required results (Vaishnavi & Kuechler, 2004). In this chapter the methods used during the course of this research project are presented.

## Research Design

According to Vaishnavi & Kuechler (2004) typically research designs commence with the identification of the problem. The initial thesis proposal was the key deliverable for this phase of the design process.

Once the problem has been identified, the researcher prepared the system for the suggestion phase. According to Vaishnavi & Kuechler (2004), it is during this phase wherein new functionality is identified based on novel configurations of new or existing systems.

### Suggestion Phase.

The identification of an IT research problem is split into a number of steps. This includes problem identification, literature research and a pre-evaluation of relevance (Offermann, Levina, Schönherr, M & Bub, 2009).

To determine the high level requirements both primary and secondary data was evaluated. The primary data was derived from answers participants gave during questionnaire

(Appendix A). The secondary data was obtained from published documents, technology evaluation and literatures relevant to the scope of the project. Literature research was also used as unsolved problems are often discussed in scientific publications and practitioner reports (Offermann et al., 2009).

Combining these methods, would allow the study to take on the combined quantitative and qualitative approach of research. This approach would therefore, enable the suggestion phase, obtain the advantages of both research methodologies and overcome their individual limitations (Offermann et al., 2009).

Based on Vaishnavi & Kuechler (2004) advice, that functionality may be identified from novel configurations of existing systems, the researcher decided to conduct an evaluation of other similar existing technologies for tuning MySQL Community Servers. The researcher found there where two solutions relevant to this research. These solutions were tuning_primer.sh and MySQLTuner.pl. The researcher decided to use qualitative research during the course of this evaluation.

The purpose of the questionnaire research was to provide the researcher with a clear understanding of issues database administrators currently experience. The questionnaire (Appendix A) was emailed to a group of twenty four database administrators working in the ICT industry in Ireland. This was used, as the main data-gathering instrument for this study. The questionnaire contained both open and closed format questions and wished to examine a number of characteristics of the participants including experience, perceptions of similar tools and requirements for the proposed tool.

The researcher decided to use quantitative research for two different purposes, the identification of high level requirements and experimental research for validation of the

framework. For the purpose of gathering high level requirements, the researcher considered a number of candidate quantitative methodologies. These included:

- Structured Face to face interview
- Structured Telephone interview
- Questionnaire via email

During the course of this evaluation, the researcher found that structured face to face interview techniques would be of benefit. It could provide the research team with the significant advantage of building a rapport with the respondent (Leedy & Ormrod, 2005).This method was later discounted due to practical considerations. The key finding was lack of project funding, meant no additional resources could be employed to assist in conducting these interviews.

Structured telephone interview techniques were also considered. This method overcame the limitations associated with face to face interviewing (Leedy & Ormrod, 2005). Telephone interviews, are cheaper to conduct and less resource intensive than the face-to-face equivalent. The researcher discounted this method, due to a realization that 'cold calling' participants may in actual fact alienate survey targets.

The third and final method considered was questionnaire. The researcher adopted this method as it was found that questionnaire would overcome the resourcing issues associated with face to face interview techniques. The researcher also found this approach was less likely to alienate participants than the telephone interview technique. According to Leedy & Ormrod (2005) questionnaire would be an inexpensive approach to access a large dispersed population. The major drawback considered before finally adopting this approach was the likelihood of a low return rate.

After identifying a problem and pre-evaluating its relevance, a solution has to be developed in the form of an artefact.

**Design Phase.**

The artefact design is a mostly a creative process, relying on the individual capabilities of the engineering team. IS literature does not provide extensive guidance on this process. After identifying a problem and pre-evaluating its relevance, a solution has to be developed in the form of an artefact. This is conducted during the development phase (Offermann et al., 2009).

**Evaluation Phase.**

According to Vaishnavi & Kuechler (2004) it is during the evaluation that the artifact is evaluated according to criteria the thesis proposal. During this phase of the research design, the researcher wished to determine if the proposed framework was successful in meeting its stated aims. To conduct the evaluation design research a number of true and pre-experimental designs were evaluated. These included

- Posttest-Only design
- Pretest-Posttest Design Control Group design
- Experimental Pretest-Posttest Design

The first method evaluated was Posttest-Only design. The researcher found this method would only be applicable when pretesting is not possible or would influence the results. The researcher eliminated this method, as pretesting is essential to determine the effect of implementing the recommendations on the MySQL community server.

Pretest-Posttest Design Control Group design was then evaluated. The researcher found that using a control group would be of little benefit as the change taken place was known and not subject to other possible explanation.

The last method evaluated was the Experimental Pretest-Posttest Design. In this method a group is made subject to a pre-experimental observation, a treatment is administered and then observed again (Leedy & Ormrod, 2005). This method was chosen.

During the Experimental Pretest-Posttest design, the researcher configured a MySQL Community server with a number of tables containing different a number of different data and indexing requirements. The group was measured prior to tuning. After tuning occurred the groups are re-measured to determine if any change in performance has occurred.

**Hypothesis Testing**

During the evaluation phase, the researcher expected to test the hypothesis identified during Chapter 1. The researcher, proposed to use the results gathered during the evaluation method, to determine if the framework satisfied the hypothesis.

Hypothesis, H1, would be proven, once the framework provided recommendations based on database workload. These recommendations would be derived, from published documents, technology evaluation and literatures relevant to the scope of the project. The researcher proposed to test this hypothesis during the Pretest phase of the Experimental Pretest-Posttest design.

Hypothesis, H2, would be proven, once the recommendations, identified by the framework, were determined to be applicable to a MySQL Community Server. The recommendations should have no adverse effect on the overall performance of the database. The researcher proposed to test this hypothesis during the Posttest phase of the Experimental Pretest-Posttest design.

**SDLC Process and Procedures**

The major System Development Lifecycle (SDLC) models used in the IT domain are

- Waterfall model
- Iterative model
- Spiral model

The classic waterfall System Development Lifecycle (SDLC) provides an easy method to manage the SDLC. In this model, each phase provides specific deliverables and a review process. According to Larman & Basili (2003) the rigidity of this is often viewed as its major disadvantage, stakeholders are only able to evaluate the software after test phase. Therefore, it is only possible to receive feedback after the software development process is complete.

Belani el al (2009) believes the iterative model solves the major criticism of waterfall. It achieves this by sub dividing the project into smaller, more manageable, units of work. This allows stakeholders, the opportunity to offer feedback at the end of iterations. This model consists of four separate phases,' inception, elaboration, construction and transition. Each phase requires different intensities of disciplines such as, business modelling and testing.

The spiral model was designed, to combine the features of the waterfall and prototyping models. This included risk assessment. The spiral model uses a cyclic approach to grow a system's degree of definition and implementation. This model also uses anchor point milestones to ensure stakeholder commitment throughout the project lifecycle.

After reviewing the three major models, the iterative model was chosen for this project. During the course of the evaluation, the researcher found the rigidity associated with the waterfall model made this approach impractical for this project.

The researcher then considered the spiral model but found this approach is not suitable for small scale projects. The iterative model was then selected as it overcomes the limitation of

the waterfall approach and is suitable for small scale projects. This thesis would represent a single iteration of the iterative SDLC.

**SDLC Key Deliverables**

The inception phase included the activities necessary, to define the project and get it underway. These activities include high-level requirements and project scope definition. These deliverables were provided in the scope section of chapter one.

According to Kroll (2001), the elaboration phase, should include the activities which prepare for software construction. This includes, developing detailed requirements, conceptual models, establishing the software development environment, constructing prototypes to evaluate technical or presentation ideas. The key deliverables, presented during the elaboration phase, were the Detailed Requirements and Implementation methodology.

During the construction phase, the emphasis shifted towards implementation. During this phase missing requirements were identified and the analysis and design models were completed. The initial system was built. At this stage, stakeholders were able offer feedback of the first iteration of the product, identifying misunderstood or misinterpreted requirements. At the end of the construction phase an executable iteration of the proposed framework was completed for evaluation. The key deliverables, presented during the construction phase are the Class Diagrams and Sequence Diagrams.

The focus of the transition phase was to prepare the software for delivery for end users. The transition phase included testing the product in preparation for release. Some minor adjustments were implemented at this phase, based on user feedback. At this point in the lifecycle, user feedback focused mainly on fine tuning the product, configuring, installing and usability issues, all the major structural issues having been worked out much earlier in the project

lifecycle. The key deliverables, presented during the transition phase are the results of the Pretest Posttest experimental design.

**Resource Requirements**

To identify the high level requirements for the framework, the researcher choose a total of 24 participants were asked to participate in the questionnaire. To obtain maximum benefit selection criteria was imposed on the survey participants. The selected participants were required to be qualified database administrators working within the ICT industry within Ireland.

For coding and implementing the framework, a single hardware environment (Appendix I) was used. The framework was then coded using the Eclipse SDK 3.5.1 environment.

To conduct the one group Pretest Posttest research, a MySQL Community Server was then installed on the hardware environment (Appendix I). To generate a synthetic workload, Apache's JMeter was then installed on the same hardware environment.

Figure 1 displays the JMeter JBDC connection configuration page. The researcher identified a number of test scenarios, to be used during the course of this research (Appendix H). JMeter was then configured to execute all scenarios concurrently. Each test scenario represented a thread on JMeter. Tests were executed for durations of 60 minutes. At the end of each test the average duration time of each thread was recorded. These measurements were then used to determine if the framework had achieved its stated aims, during the transition phase of the project. The results recorded using JMeter during Pretest are provided in Appendix J. The results recorded using JMeter Posttest is provided in Appendix L.

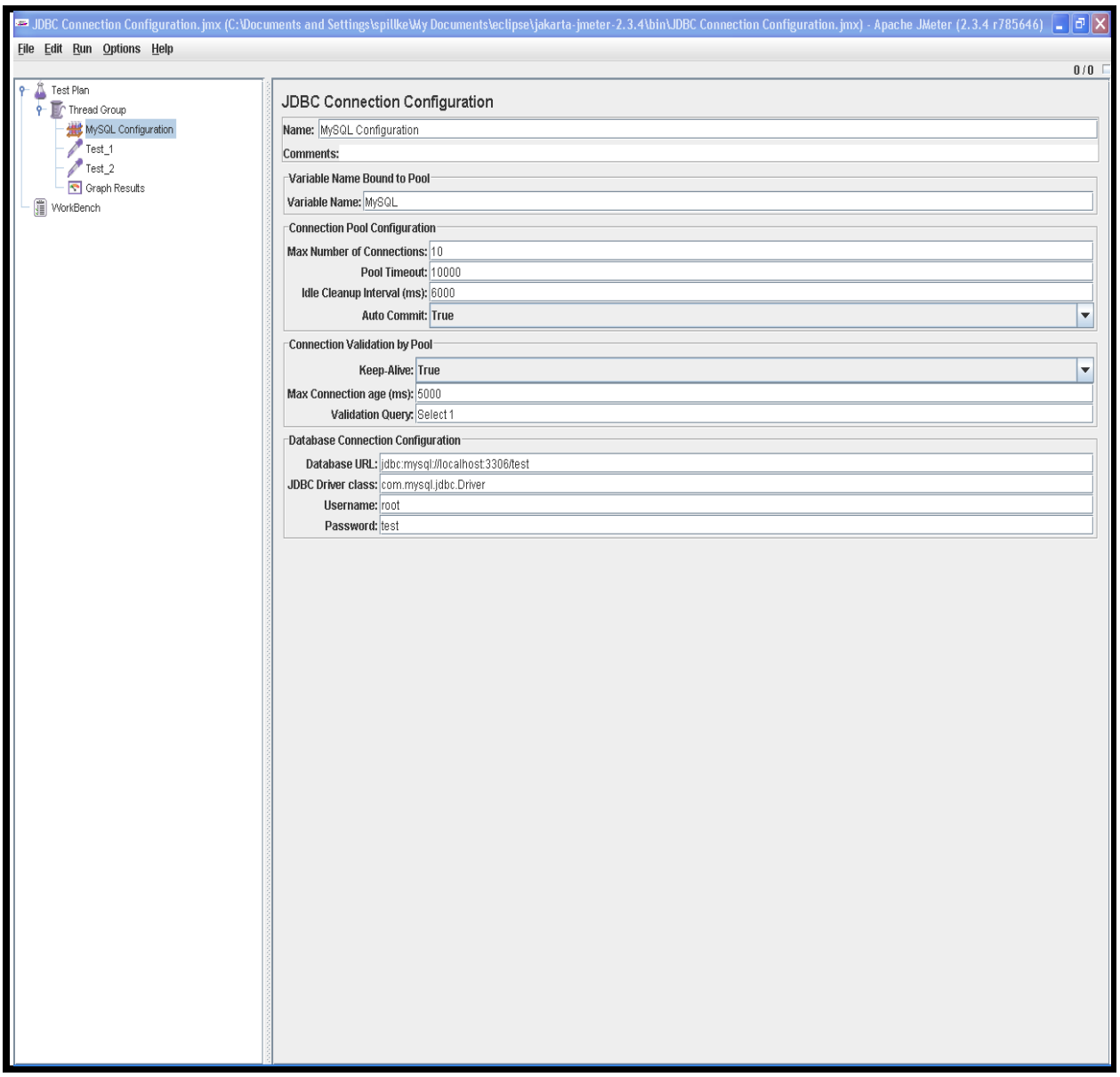



*Figure 1*. JMeter MySQL Configuration

**Chapter Summary**

This chapter identified the research and development methodologies used during the course of this thesis. Quantitative research methods were chosen, to gather high level requirements and determining the effect of applying recommendations on the MySQL Community Server. Qualitative research methods were chosen to evaluate existing technologies.

The iterative SDLC was chosen and key deliverables identified. Finally this chapter concluded with a review of the resource required to conduct this research.

The next chapter contains the results and an analysis of the research executed during the course of this project.

# Chapter 4 – Project Analysis and Results

## Introduction

In chapter one, the researcher stated this thesis would solve the problem of providing an automated database tuning tool, which will be used to solve the problems of identifying effective recommendations for index and system variable selection in MySQL community servers using scientific processes and repeatable techniques.

This chapter contains a number of results, key to solving this problem. The chapter commences, with an analysis of tools currently used to tune MySQL community databases. The results of the questionnaire research are then presented and analysed. The chapter concludes, with a presentation of the key deliverables for the SDLC.

## Review of Existing Solutions

During the literature review, Annesley (2006) stated, that a MySQL Community server does not contain the automated tuning capabilities of its commercial rivals. Despite, this lack of support for automated tuning, MySQL Community server remains one of the most popular open source databases available in the world today. This has resulted in database administrators, or indeed enthusiastic users, developing tuning solutions with varying degree of success. During this section, reviews of the two most popular solutions are presented. These solutions are tuning_primer.sh and MySQLTuner.pl

The researcher determined both solutions identified for this research have a number of common characteristics. They are open source products, freely available for download on the internet. Neither solution has associated documentation. Both solutions have user communities who for the large part communicate through internet blogs.

The researcher determined the data required from this study could be collected from a review of source code. The study objective was to investigate how these technologies approached the problem of identifying recommendations for tuning a MySQL community server.

The analysis was of the source code was made under the following headings

- General Characteristics
- Index Recommendations
- System Variable Recommendations

**MySQLTuner.pl.**

MySQLTuner.pl is script which is free to download on the internet. This script is written in the PERL (programming extraction reporting language) programming language.

The purpose, of this script is to analyze performance of the MySQL database server. Then based on the statistics gathered, the tool provides recommendations on system variables which are suitable candidates for adjustment.

The researcher found, MySQLTuner.pl has a number of similarities to tuning_primer.sh. Firstly, it is a read only script, which presents the database administrator with an overview of the MySQL server's performance and similar to, tuning_primer.sh shell script MySQLTuner.pl provides recommendations based on the statistics it gathers.

### *General Characteristics.*

The researcher found MySQLTuner.pl contained the following general characteristics.

| | | |
|---|---|---|
| High level programming language | : | Perl |
| Operating System Support | : | Linux, UNIX |
| MySQL Support | : | 3.43, 4.0, 4.1, 5.0, 5.1 |
| Configurable | : | No (all decisions are hard coded) |

### *Indexing Recommendations.*

MySQLTuner.pl, does not issue indexing recommendations on individual tables. It does provide a percentage of the total queries run that were slow. It achieves this, using the slow queries and Questions statistics from the SHOW STATUS command. Appendix D contains typical output from running the SHOW STATUS command.

### *System Variable Selection.*

Most of the recommendations, made by MySQLTuner.pl, are related to System Variable configuration. This is where the strength of this tool resides. It commences, by determining what storage engines are enabled on the MySQL sever. If the script finds a storage engine isn't being used, it recommends that engine be disabled.

Using a number of calculations, based on information extracted from the SHOW STATUS command, detailed recommendations are provided on memory usage and resource allocation MySQLTuner.pl, provides recommendation on a number of different variables, including

- User connections
- Query Cache
- Thread Cache

- Table Cache
- Table locks
- Percentage of operations which required sorts
- Percentage of join operations on non indexed fields

**tuning_primer.sh.**

The tuning primer script (tuning_primer.sh) was developed by Matthew Montgomery (Montgomery, 2009). The purpose of this shell script is to optimize MySQL installations, by offering recommendations on a number of key configurations. It supports multiple releases of MySQL server.

The only condition Montgomery specifies is the MySQL server must run at least 48 hours before the first run of the Performance Tuning Primer Script, to be efficient (Montgomery, 2009). Once this condition is met, this script is executed and then provides tuning recommendations for MySQL database servers.

According to Montgomery (2009), this script provides recommendations for the max connections, worker threads, key buffer, query cache, sort buffer, joins, temp tables, table cache, table locking, table scans (read_buffer) and innodb status (Montgomery, 2009).

The tuning_primer.sh script, has received wide spread support amongst the open source community. It is generally taught to be valuable in MySQL optimization process. However, similar to the MySQLTuner.pl script, the database administrator provides no input into how recommendations are determined.

***General Characteristics.***

| | | |
|---|---|---|
| High level programming language | : | BASH |
| Operating System Support | : | Linux, UNIX |

| | | |
|---|---|---|
| MySQL Support | : | 3.43, 4.0, 4.1, 5.0, 5.1 |
| Configurable | : | No all decisions are hard coded |

***Indexing Recommendations.***

tuning_primer.sh does not issue indexing recommendations

***System Variable Selection.***

Using a number of calculations based on information extracted from the SHOW STATUS command, detailed recommendations on memory usage and resource allocation are made. Recommendations, provided at this stage include

- User connections
- Query Cache
- Thread Cache
- Table Cache
- Table locks
- Percentage of operations which required sorts
- Percentage of join operations on non indexed fields

**Results of Analysis.**

As a result of this analysis, the researcher made a number of findings relevant to this project.

- Both scripts provide support for all major releases of MySQL server
- Both scripts only support Linux and Unix operating systems
- Both scripts make no recommendations on indexing
- Both scripts make extensive System Variable selection recommendations

- All recommendations made are hard coded within the scripts

**Questionnaire Results**

Completed responses to the survey were received from 18 individuals with experience of working as DBA's within Ireland. This section contains the results of the questionnaire.

To gauge the level of interest, the questionnaire asked "How interested would you be in using JMySQLTune, an automated tuning product which improves performance on an open-source MySQL community server?"

**Interest Level in proposed product.**

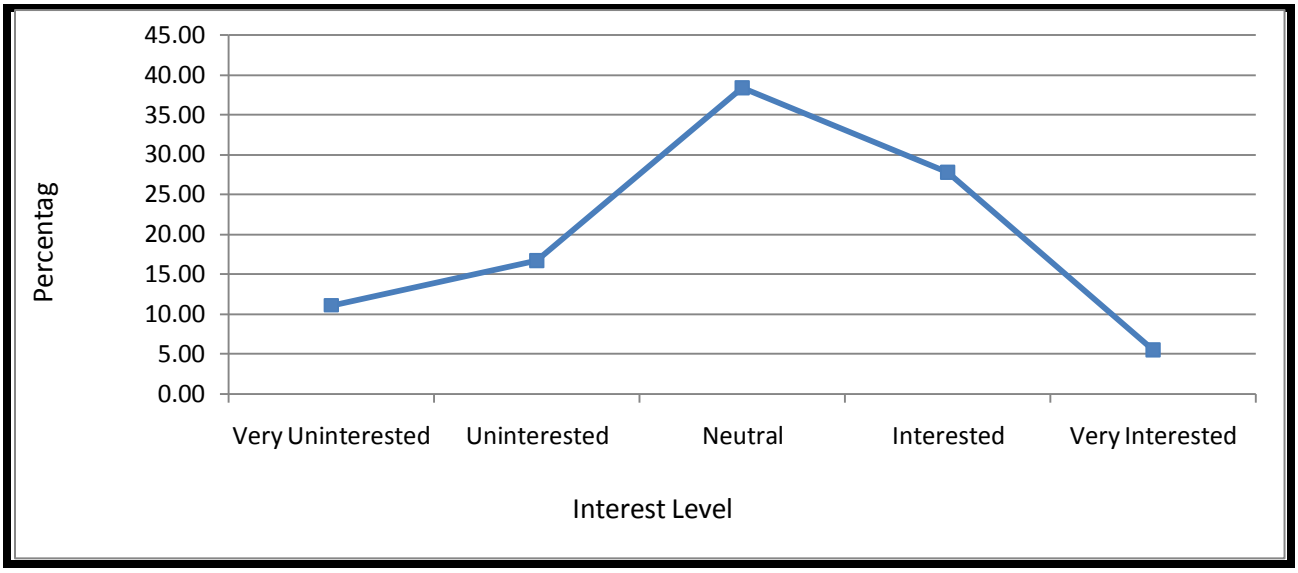



*Figure 2*. Question 1 Result

Figure 2 demonstrates the results of question 2 in a normal curve. There was some demand amongst database administrators, for this product. Over 33% of participants expressed some level of interest in the proposed product. 25% of participants indicated no interest in the product.

**Experience of Participants.**

To gauge the level the experience of the participants, the questionnaire asked "How many years experience have you gained working with the following vendors databases?

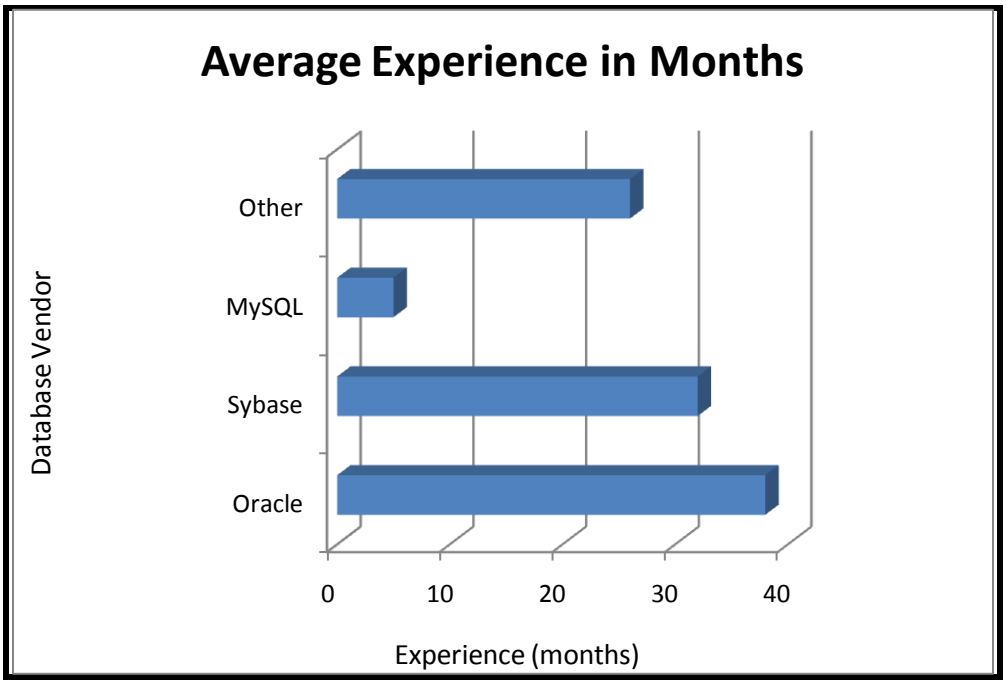



*Figure 3*. Question 2 Results

Figure 3 demonstrates the mean average experience in months the respondents had, working with relational databases from different vendors. This diagram demonstrates, amongst the participants, the average number of months working with MySQL is a fraction of other databases such as Oracle or Sybase.

**Required Feature Set.**

To gauge which features would be most desirable in to be included in the framework, the questionnaire asked "With regards the following features, Please give a ranking from 1 – 8. One being most important".

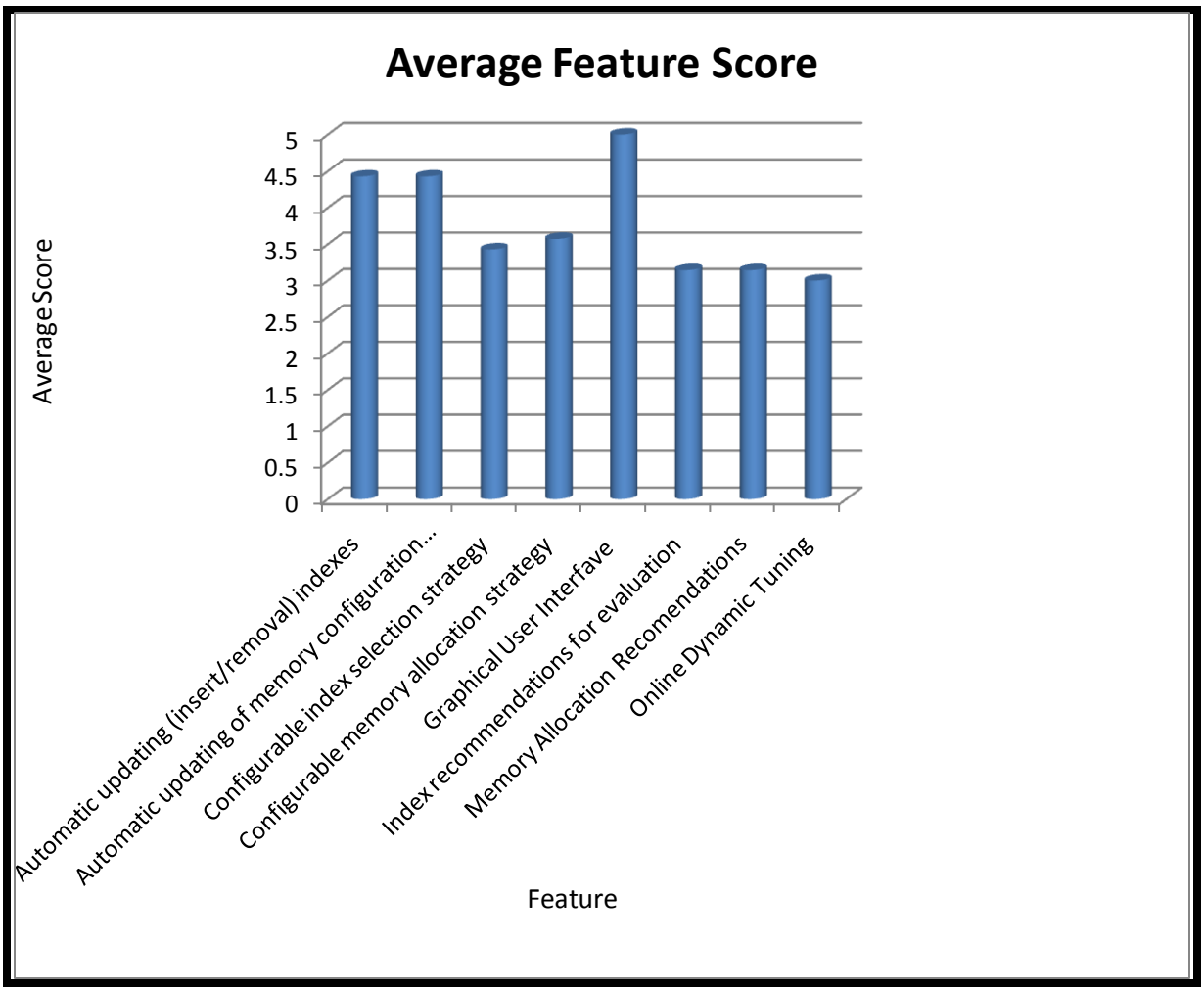



*Figure 4*. Question 3 Results

Figure 4 demonstrates the mean average score (maximum score is eight), allocated to each feature by the respondents. The higher the average score, the less desirable the proposed feature was to respondents. With an average of five, database professionals considered a Graphical User Interface to be the least important of the suggested functionality. This was closely followed by automatic updating of index or system variables.

The most important feature to the participants was online dynamic tuning followed closely by memory and index recommendations. There was also a positive reception for making these recommendations configurable.

**Index Tuning Advisor Usage.**

To gauge whether the DBAs typically use index tuning advisors, the questionnaire asked "How likely are you to use an automated tuning advisor to obtain index recommendations for a database server?"

Table 1. *Participants Interest in an Index Tuning Advisor*

| | Percentage of Participants interest in Index Tuning Advisor |
|---|---|
| Very Interested | 14.28 |
| Interested | 28.57 |
| Neutral | 42.57 |
| Uninterested | 14.28 |

Table 1 presents the results of Question 4 from the questionnaire. Over 42% of participants offered a positive response to this question. The option which received the most support was the neutral response.

**Cache Tuning Advisor Usage.**

To gauge whether the DBAs typically use cache tuning advisors, the questionnaire asked "How likely are you to use an automated tuning advisor to obtain index recommendations?"

Table 2. *Participants Interest in a Cache Tuning Advisor*

| | Percentage of Participants Interest in a Cache Tuning Advisor |
|---|---|
| Very Interested | 14.28 |
| Interested | 57.14 |
| Neutral | 14.28 |
| Uninterested | 14.28 |

Table 2 presents the results of Question 5 from the questionnaire. Over 71% of participants offered a positive response to this question. The option which received the most support was the interested response.

**Usage levels of the proposed tool.**

To gauge whether the DBAs would use the proposed product, if it was available, the questionnaire asked "If this product were available today, would use it for tuning MySQL community servers?"

Table 3. *Percentage of Participants Interested in using the proposed Product*

| | Percentage of participants interested in using the proposed product |
|---|---|
| Interested | 42.85 |
| Neutral | 57.15 |

Table 3 presents the results of Question 6 from the questionnaire. Over 42% of participants offered a positive response to this question. The option which received the most support was the neutral response. No respondent declared themselves to be very interested or indeed very uninterested in using the proposed product.

**Suggestions for additional features.**

To gather the DBAs opinions on what they would require from the tool, the questionnaire asked to respond in plain text to the following question. "Are there additional features, you feel should incorporated into this product?" Of the 18 questionnaires 12 people answered this question.

- The majority of participants stated that they would only use such a tool for advice. They do not want the tool to automatically implement any change on the database

- One respondent suggested advice should be based on historical statistics contained within the database
- One respondent suggested the poor SQL coding is the true source of bad performance in many databases and therefore should be also examined

**Experience of other Automated Database Tuning Tools.**

To gather the DBAs opinions on similar tools which they may have used the following question was asked. "Please indicate the degree to which you agree/disagree with the following statements about your experience of automated database tuning products?"

Table 4. *Participants Experience of other Automated Database Tuning Tools*

| | Percentage Agree | Percentage Neutral | Percentage Disagree |
|---|---|---|---|
| They mostly improved performance of indexes | 33.3 | 48.6 | 16.2 |
| They mostly improved performance of memory | 0 | 83.2 | 16.2 |
| They were configurable | 33.3 | 48.6 | 16.2 |
| They improved the overall system performance | 16.2 | 66.6 | 16.2 |
| They were user friendly and easy to use | 33.3 | 33.3 | 33.3 |

Table 4 presents the results of Question 7 from the questionnaire. In all five questions, the neutral option received the highest proportion of responses. No respondent agreed that in their experience an automated database tool improved the performance of database memory.

**Current Method of Tuning MySQL Community Databases.**

To determine how the DBAs currently tune MySQL Community Servers the following question was asked "How would you currently tune indexes and/or cache memory in a MySQL community server?"

Table 5. *Current Method of Tuning MySQL Community Databases*

| | Percentage |
|---|---|
| I don't know I have never tried | 45 |
| Using a Perl script I downloaded from the internet | 5 |
| Using manual intervention | 50 |

Table 5 presents the results of Question 8 from the questionnaire. Only five percent of respondents had experience of using existing products for tuning MySQL community servers. Fifty percent of respondents had used manual intervention.

**Comments and suggestions.**

Finally the DBA were asked to enter in plain text any other comments they may have. Of the 18 questionnaires 2 people provided a reply to this question.

- One person stated that he had no experience of MySQL Community Server
- One person stated that in order to make accurate recommendations it was critical to take workload into consideration.

**Elaboration Phase Deliverables**

After analysing the results of primary and secondary research the following detailed requirements were identified

**Functional Requirements Identified from primary research.**

After establishing the experience of the participants, the following functional requirements were identified as a result of the primary questionnaire research

- The framework should operate in an online tuning manner
- The framework should make recommendations on system variable configurations
- The framework should make recommendation on index selection
- The framework should provide the database administrator with the option to offer input into the index selection process
- The framework should provide the database administrator with the option to offer input into the cache tuning process

**Functional Requirements Identified from secondary research.**

During the literature review the following functional requirements were identified

- The framework shall use a decision tree approach to make recommendations
- The framework shall use MySQL System Status command to determine the status of the systems cache memory
- The framework shall use Slow Query log to determine which SQL statements are candidates for indexing recommendations.
- The indexing decision tree will be based on a heuristics approach suggested by Bujdei et al., 2008
- The System variable decision tree will be based on the approach to system variable tuning suggested by (Kofler, & Kramer, 2005, Cabral & Murphy, 2009)

**Non Functional Requirements.**

The following non functional requirements were identified

- The tool will be developed using the JAVA programming language.
- The tool will communicate with the MySQL Community Server via Java Database Connectivity (JDBC)
- The decision tree shall be configurable via XML files

**Framework Overview.**

During chapter one, the researcher stated the purpose of the thesis, to be the provision of a framework capable of providing recommendations for index and system variable selection in MySQL community servers. During this section, the researcher provides an outline description, of the proposed framework. The researcher details, how this application provides both system variable and index recommendation for MySQL database servers. The researcher offers some justification, as to why this architecture was chosen. The emphasis of this evaluation is to determine whether the proposed solution provides a suitable replacement for the existing technologies and to ensure that all of the initial problems were resolved.

The researcher developed JMySQLTune, using the JAVA programming language. The Java Database Connectivity (JDBC) allowed JMySQLTune query and if so required update the MySQL Community server database using the Structured Query Language (SQL) (Konchady et al., 2008). The researcher proposed JMySQLTune, would examine the slow query log to determine, which if any existing queries are causing poorest performance within the database. The physical tuning policies, would be maintained to two separate files Index.xml and SystemVariable.xml

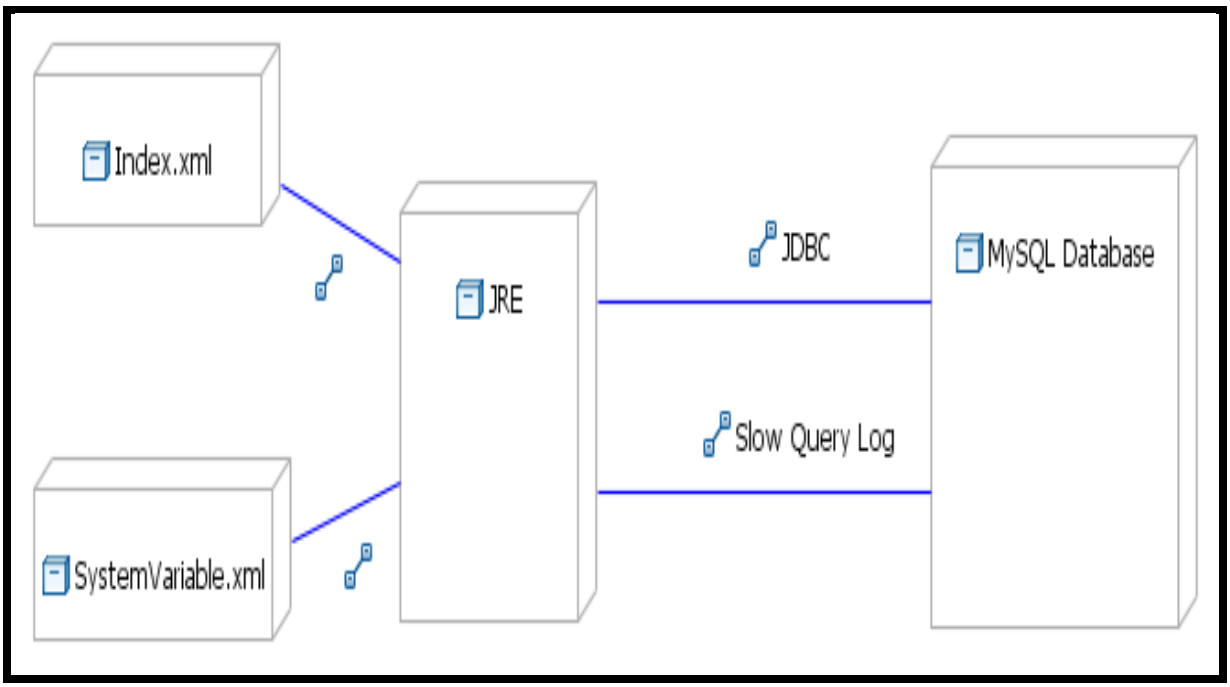



*Figure 5*. JMySQLTune Architecture

Figure 5, provides an overview of the proposed frameworks architecture. JAVA is a high level object oriented language, developed by Sun Microsystems. Using Java, as the programming language increased the proposed frameworks portability, as this is platform independent. Java applications run within a Java Runtime Environment (JRE). This Java Runtime Environment is an emulator program, which sets aside part of the hard drive to allow the Java Virtual Machine to execute. Java JRE's are available for all major operating systems (Burd et al., 2006). Developing JMySQLTune with Java and JDBC will result in a product which is both platform and vendor independent, allowing it to run on a PC, a workstation or a network computer (Burd et al., 2006).

To analyse indexing and system variables, JMySQLTune adopted the decision tree approach presented by Bayan & Cangussu (2005). The ability to reliably identify the source of performance bottleneck, within a particular MySQL database server accurately is crucial.

Inaccurate analysis will result in recommendations which are likely to result in performance degradation.

Decision trees provide a relatively simple and easily understood approach to problem diagnosis (Mousa et al., 2004, Gorea et al., 2006 and Bayan & Cangussu, 2005). Indeed, the decision trees ability to offer a hierarchical approach to decision support making methods means it is often employed in many diverse fields such as medical diagnosis, risk assessment, banking applications, strategy games, policy assessment and expert systems (Gorea et al., 2006).

Secondly, it is easy for the database administrator to understand how decisions are made and recommendations are arrived at (Bayan & Cangussu, 2005). The strength of a decision tree lies, in its ability to "analyze decision alternatives in a systematic, chronological way and provide an easy to read graphical presentation to decisions under considerations." (Mousa et al., 2004 p1268 p2).

The decision tree will be provided via Extensible Mark-up Language (XML). This simplistic approach will provide experienced database administrators, with the ability to manually configure the index selection and/or cache configuration policy, as to meet their individual requirements, or the requirements of the organization in which they reside. The decision to allow database administrators configure the tool, if they so desire, is based on both feedback from the questionnaire and insight gained during the literature review. Many researchers acknowledge, that the database administrator adds value to performance tuning, through localized expertise gained from working with the database on a day to day basis (Chaudhuri & Narasayya, 1998, Bruno & Chaudhuri, 2006 and Comer et al., 2008). Making JMySQLTune configurable emphasizes the importance of the database administrator in the tuning process.

The XML format was chosen for as the configuration file for two reasons. Firstly, it provides a strong framework for the storage of decision trees. Secondly, in the last number of years it has become a universal standard for information exchange amongst applications (Gorea et al., 2006). JMySQLTune divides the tuning process, into two separate tasks, the index selection task and the system variable selection task. There will be two separate XML decision tree files which will reflect this structure.

The recommendations for tuning from both XML files will be derived from best practices identified. It was noted during the literature review, this approach is a commonly applied methodology in the area of physical database tuning (Chung & Hollingsworth 2004, Bruno & Chaudhuri, 2005 and Bujdei et al., 2008).

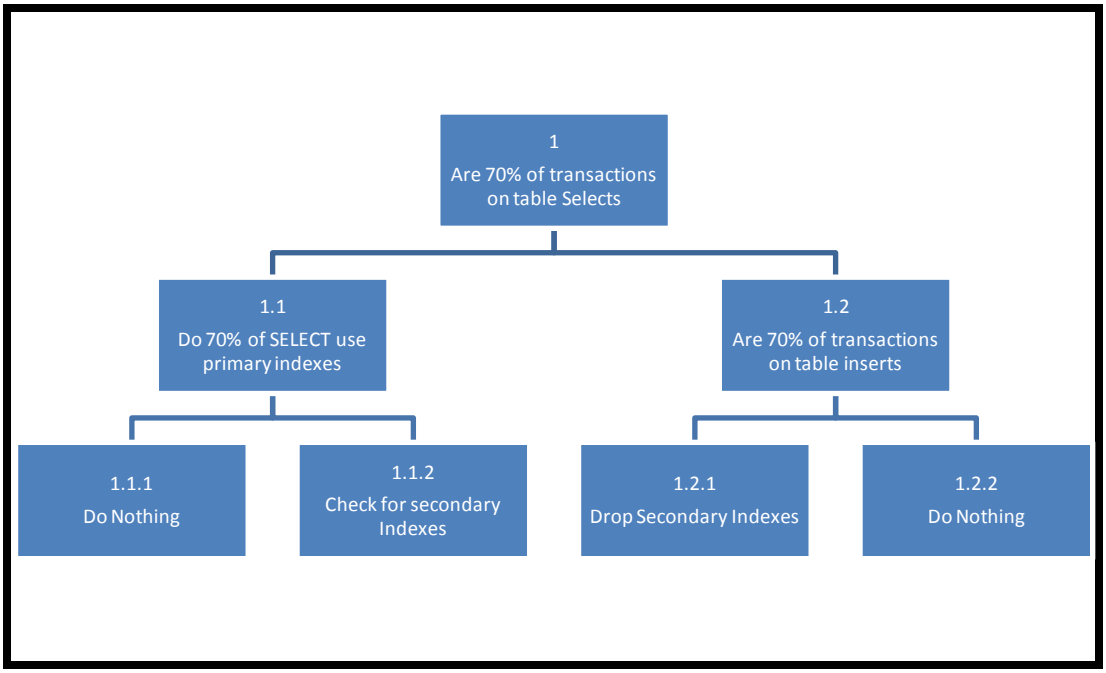



*Figure 6*. JMySQLTune Sample decision tree

Figure 6 displays a sample decision tree. In this example the XML decision tree is telling JMySQLTune to recommend dropping secondary indexes on a particular table, if over 70% of the transactions on that particular table are insert operations. If 70% of the transactions are

selects and uses primary indexes, then this XML file instructs JMySQLTune to do nothing. This is an example of the type of heuristic being adopted by this thesis.

Although, it was noted during the literature much of the latest research has revolved on-line tuning, JMySQLTune will use the more traditional offline approach. Indeed on-line tuning received most support in question 3 of the questionnaire research. The availability requirements of the open source database, such as MySQL Community server, should not be such, that scheduled maintenance windows cannot be arranged. Therefore choosing the offline approach in this case, not only complies with Sockut & Iyer (2009) rule of thumb, but is a more technically feasible approach.

**Tuning Process.**

In this section, an overview of the tuning process is presented. During the literature review, this thesis evaluated both off-line and on-line approaches to database physical tuning and noted that although many database vendors in the future wish to adopt an on-line approach, today it is more advisable to adopt the off-line method (Zenmyo, 2009, Schnaitter et al., 2009).

During regular maintenance windows, JMySQLTune will evaluate the current configuration by gathering database performance information from the MySQL database server. The 'SHOW STATUS', 'SHOW TABLE STATUS' and 'EXPLAIN', SQL commands will be used to extract performance information from the database. The 'SLOW QUERY LOG' file will also be analysed, to determine what operations are resulting in performance bottlenecks.

This information will be correlated and using the XML decision trees recommendations, will be displayed to console, for consideration by the database administrator. The database administrator can then decide to either ignore or implement these recommendations.

Weikum el al (2002), tells us the tuning process consists of a number different phases, prediction, stability and reaction. During the prediction phase, the framework assesses possible changes anticipating performance improvement. In the stability phase, it should be determined if the change has produced degradation in any other part of the system. Finally, in the reaction phase, it is determined whether to commit or rollback the changes.

As stated in chapter three, the researcher chose an experimental Pretest-Posttest experimental design, for the purposes of hypothesis testing. In the case of JMySQLTune the prediction phase occurred after pre-testing. The stability phase occurred after post-testing. From the results of post-testing, the researcher determined if the changes should be committed or removed from the system, this is known as the reaction phase.

The JMySQLTune XML index selection file was formulated using Bujdei el al (2008) heuristics based approach. Based on the results of these two tests, recommendations were displayed to console. The researcher then determined which if any index configurations need to be altered.

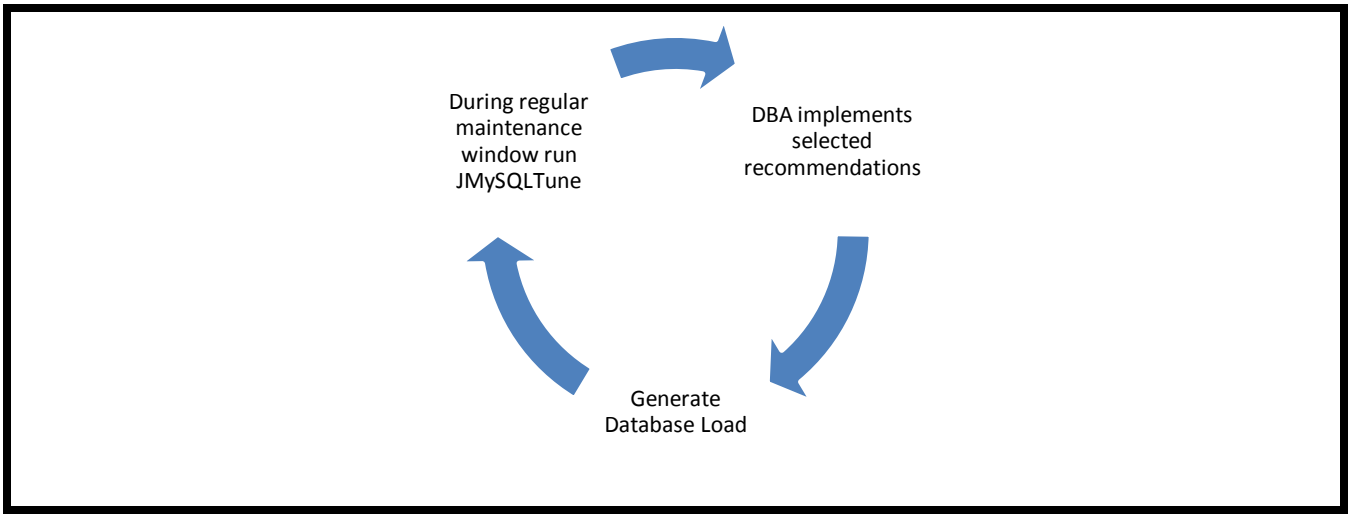



*Figure 7*. JMySQLTune Iterative Tuning Process

Figure 7 depicts the tuning process adopted by JMySQLTune, with regular scheduled maintenance windows required. The database administrator selects the relevant recommendations and then implements changes.

**Construction Phase.**

The key deliverables identified before this project could leave the construction phase were for the Java coding of the framework to be completed. Appendix M contains the flow chart for the main section of the JMySQLTune program. Appendix N contains the relevant source code for this portion of the program.

Throughout the development of this framework object oriented techniques were adopted. Class diagrams are another output of the design phase. Building a class model involves identifying the classes that should exist in the system and how they relate to each other.

Figure 8 depicts the class diagram for the XML decision portion of the application In this case business rules related to index selection are read in an XML file named index.xml. All business rules related to system variable selection are defined in an XML file named SystemVariable.xml. Upon initialization of the system, the XML Decision Tables and the Entity Description Dictionaries are loaded into Java Structures. The Decision Tables are converted into binary trees, with condition nodes defining the branches in the trees and Action Nodes defining the leaves.

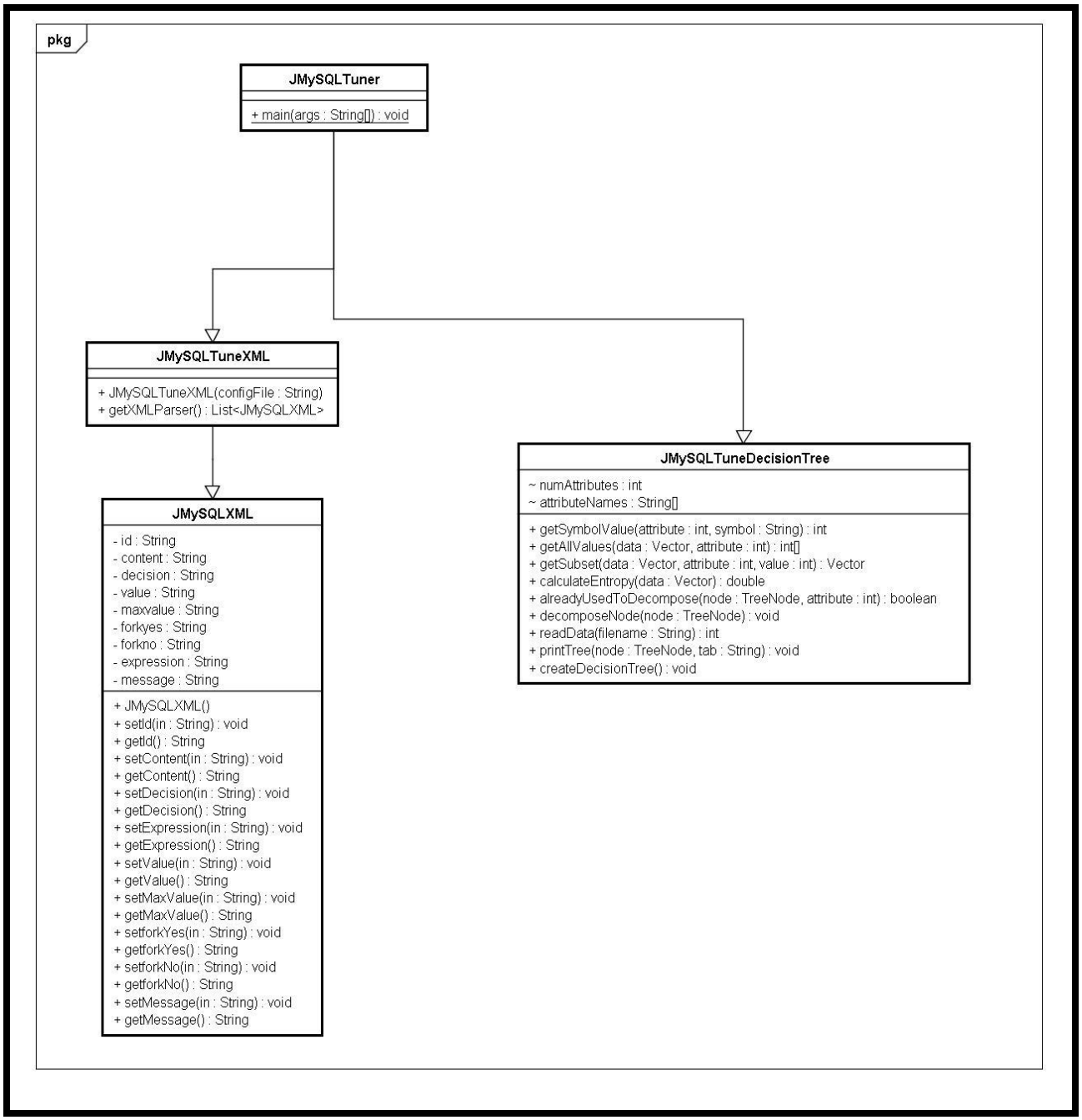



*Figure 8.* JMySQLTune Class Diagram (XML portion)

Figure 8 depicts the class diagram for the MySQL XML portion of the application. These classes parse the XML files to determine recommendations.

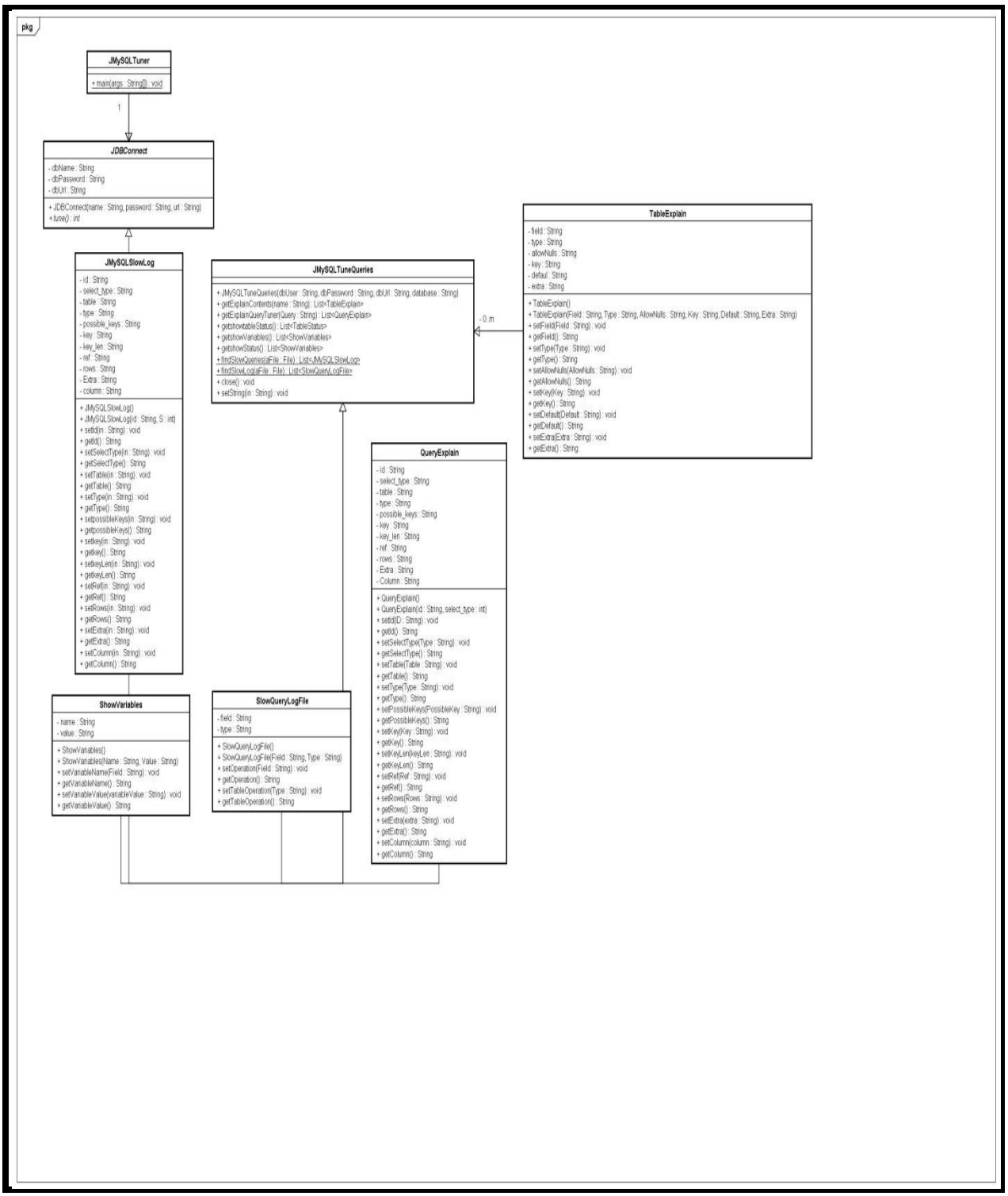



*Figure 9*. JMySQLTune Class Diagram (Tuning Portion)

Figure 9 depicts the class diagram for the MySQL communication portion of the application. These classes allowed the framework to communicate with the MySQL community server via JDBC. It also provided JMySQLTune the ability to parse MySQL Community Server slow query log file.

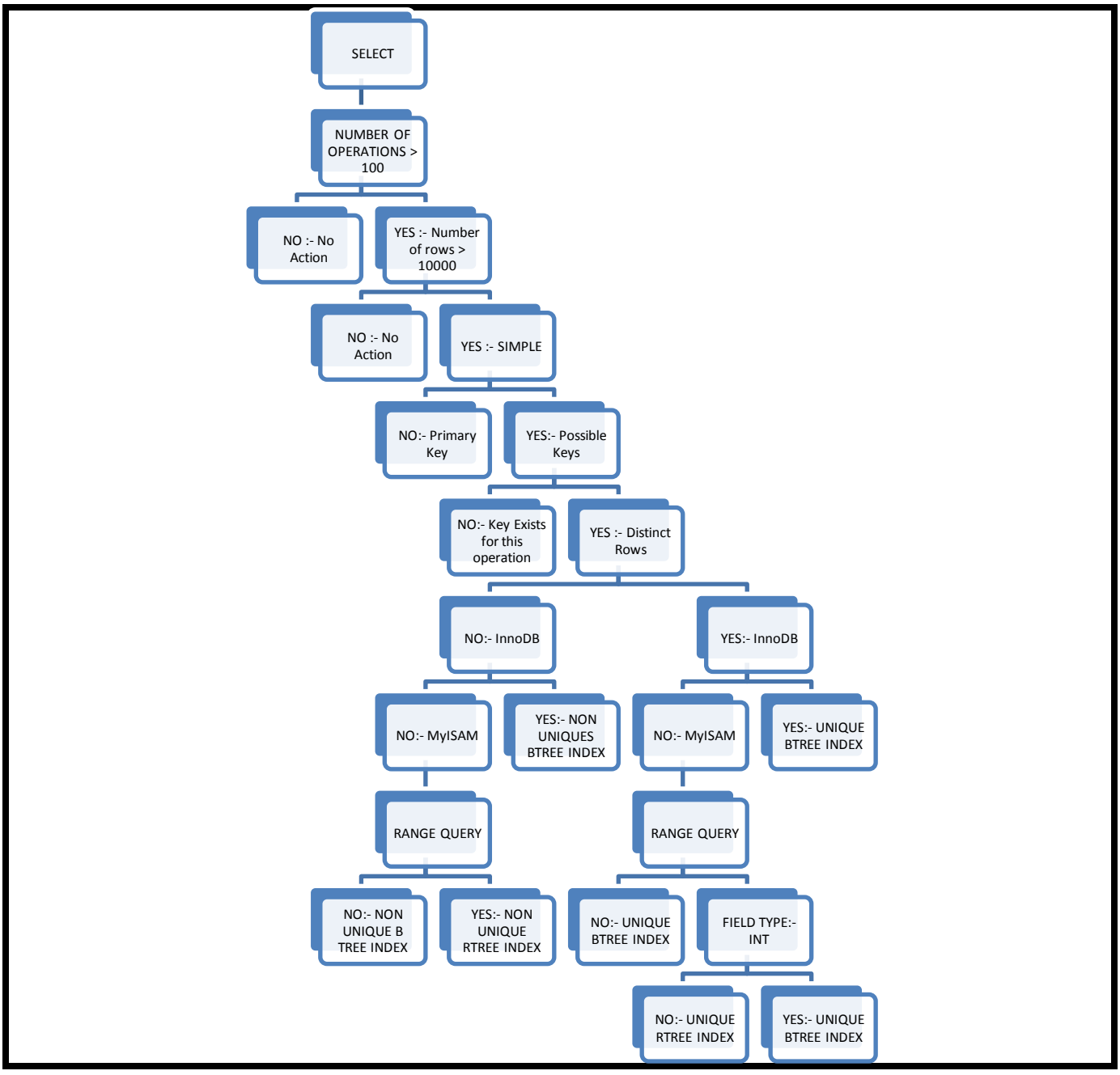



*Figure 10*. Index.xml

Figure 10 depicts the Index.xml file used for making indexing related recommendation. This XML file used a decision tree approach to determine recommendations for the index selection.

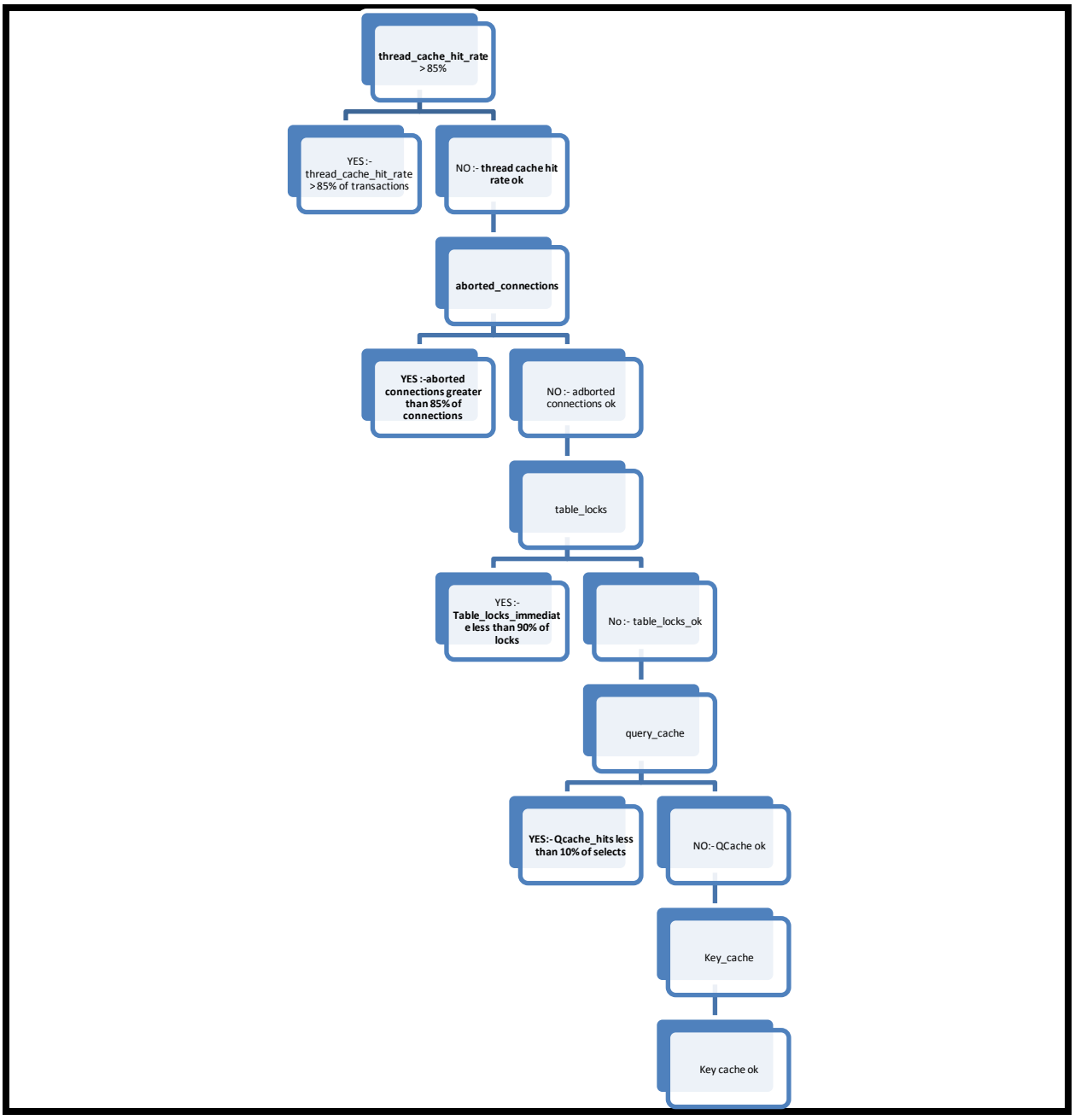



*Figure 11*. SystemVariable.xml

Figure 11 depicts a sample of the decisions made in the SystemVariable.xml file. This XML file used a decision tree approach to determine recommendations for system variable selection.

**Transition Phase**

During the transition phase, the key deliverable is the results of the pre-test post-test experimental design analysis. Appendix G contains the pre-test configuration of the MySQL Community server. A limited set of 18 test conditions (Appendix H) were identified for execution during this phase of the project.

**Pretest Results.**

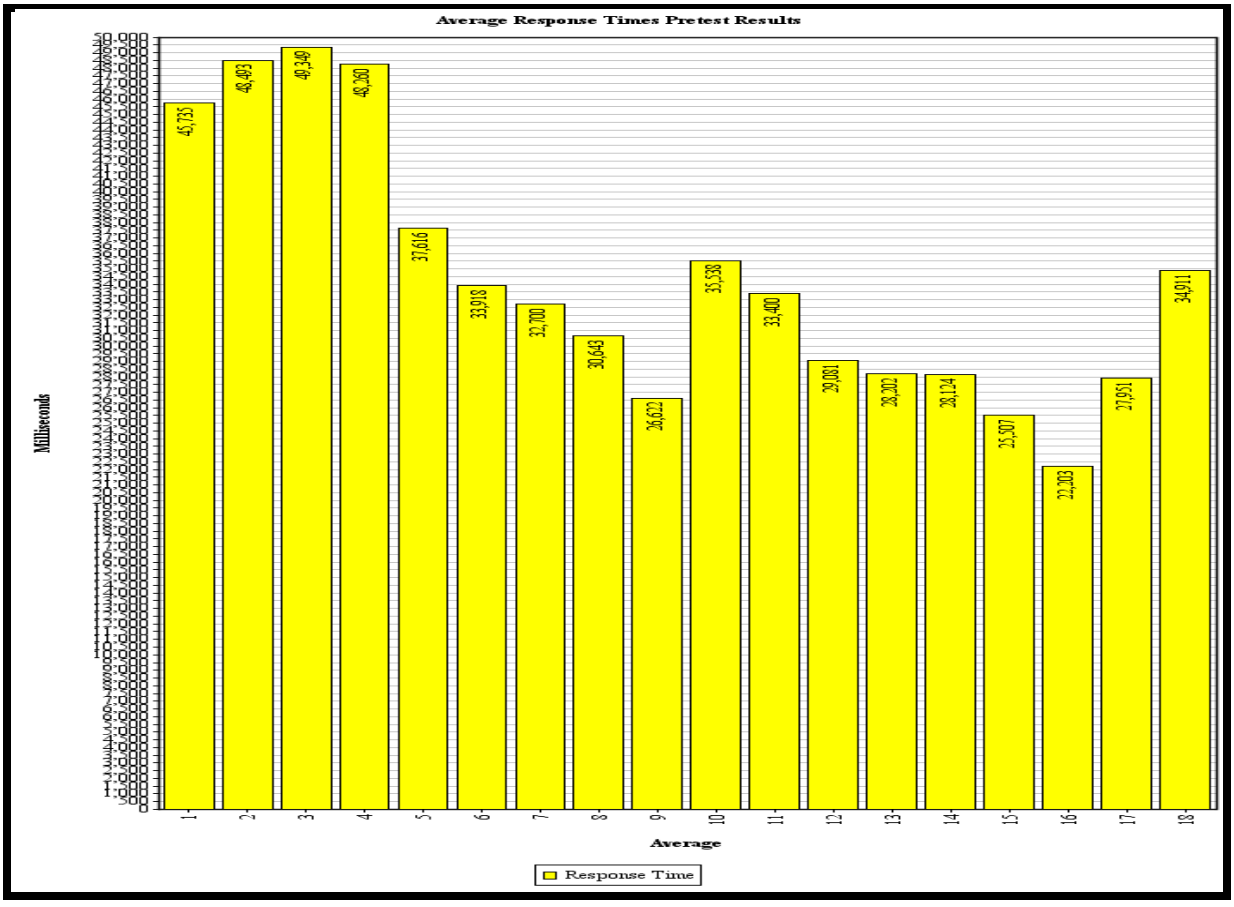



*Figure 12*. Pre-test Response time in milliseconds of SQL Queries

Figure 12, contains the average pre-test response time results (in milliseconds) for the test scenarios which were executed (Appendix H). A detailed breakdown of response times is provided for each scenario is provided in Appendix J.

**JMySQLTune Recommendations.**

After running these operations on the database JMySQLTune framework was executed, using the system variable XML System Variable selection configuration (Appendix B) and the XML Index selection configuration (Appendix C).

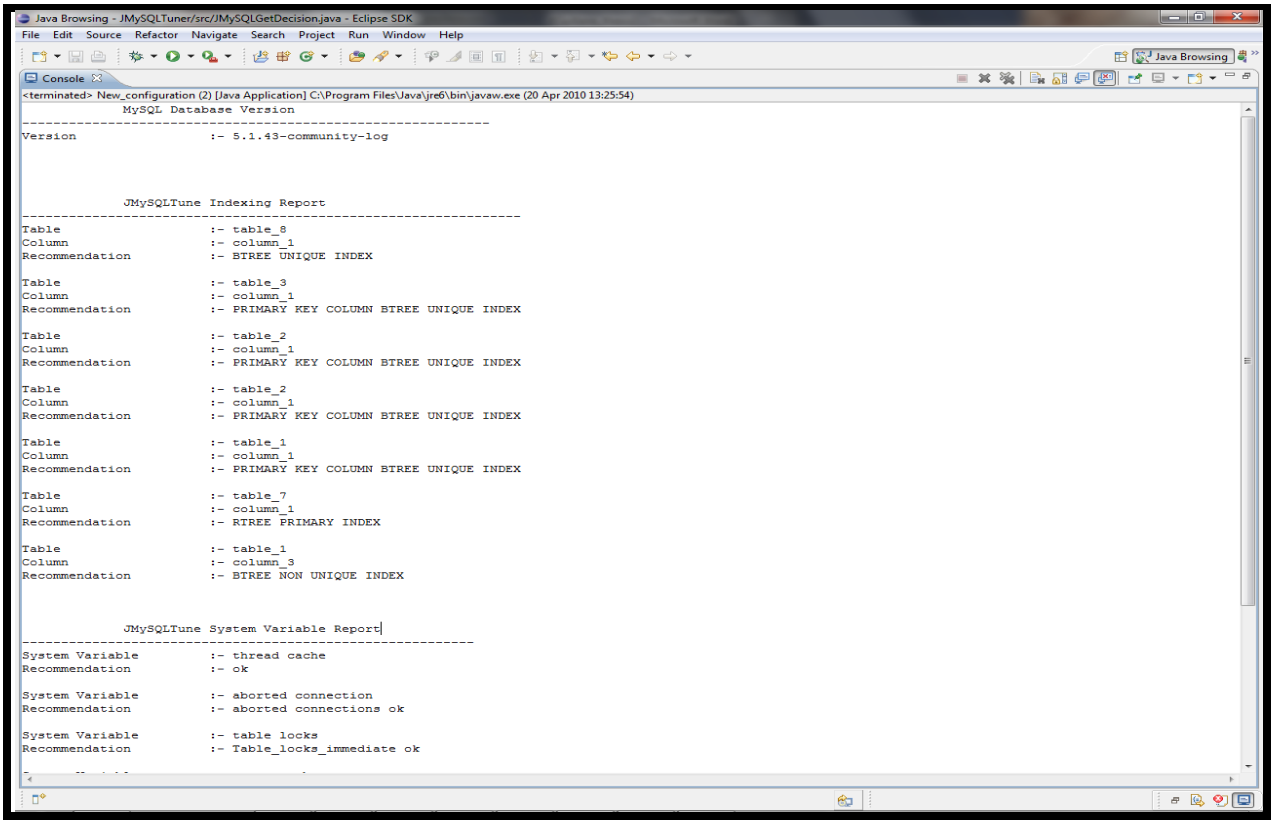



*Figure 13*. JMySQLTune Console

Figure 13, contains a snapshot of the results printed to JMySQLTune console. Appendix K, contains all recommendation outputted to console after executing the JMySQLTune tool. All recommendations were implemented and the MySQL log was emptied. All tests scenarios were then repeated. Appendix L contains the post-test response time for the updated database.

**Post-test Results.**

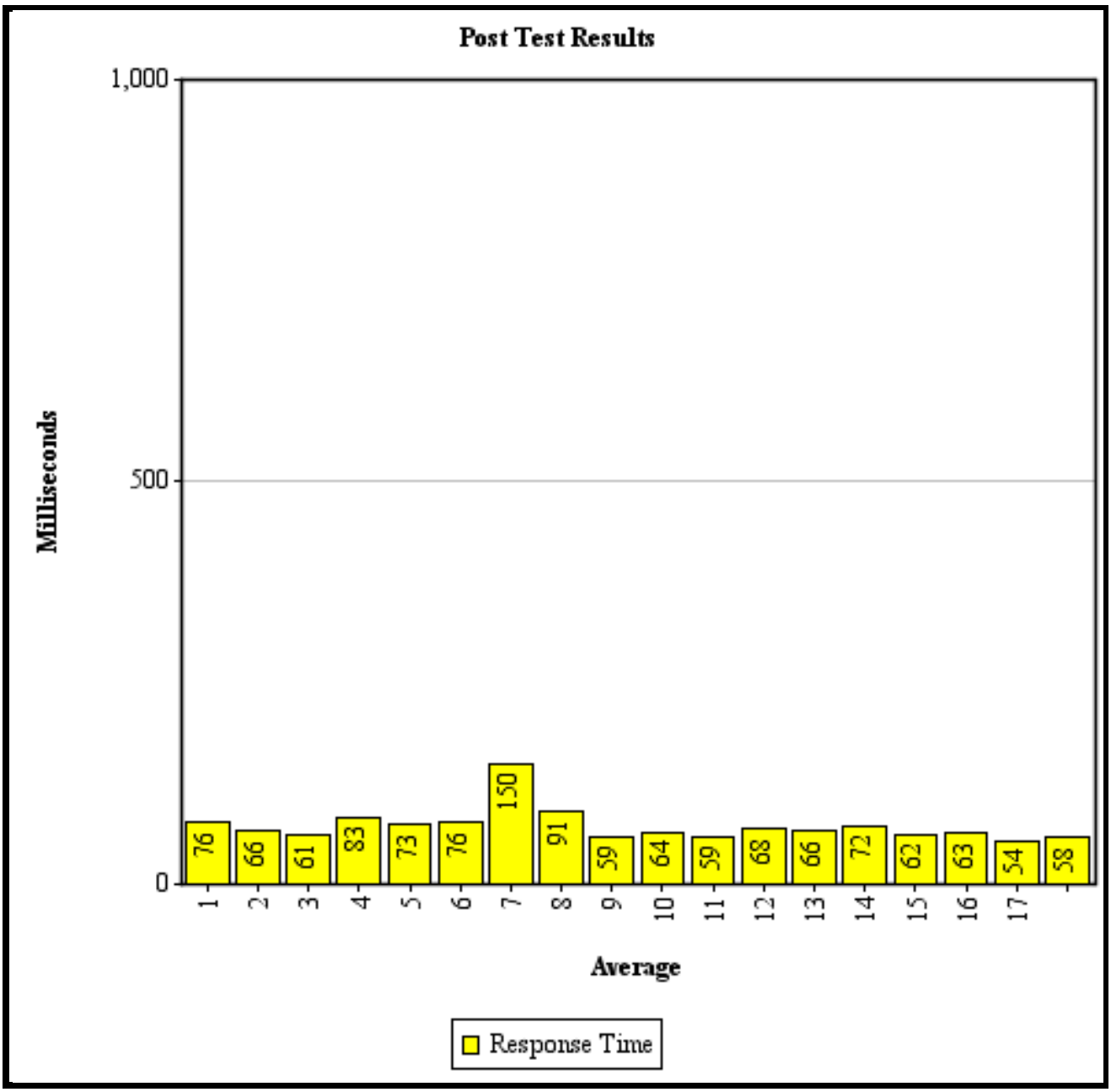



*Figure 14*. Post-test Response time in milliseconds

Figure 14, contains the average post-test response time results (in milliseconds) for the test scenarios which were identified in Appendix H. A detailed breakdown of response times is provided for each scenario is provided in Appendix L.

## Conclusion

During the course of this chapter, a number of key deliverables was presented. These deliverables represent key milestones in the project. The following chapter provides a detailed discussion of these key deliverables.

# Chapter 5- Discussion

## Introduction

This goal of this thesis was to provide an automated database tuning tool, used to solve the problems of identifying effective recommendations for index and system variable selection, in MySQL community servers..

In this chapter, the researcher will discuss the key results, presented in chapter five. The researcher used these results, to determine how successful the project has been. A brief discussion of the how the overall project was managed, also occurs. Finally, the researcher identifies some limitations and lessons learned, during the course of the project.

## Summary of Questionnaire Research results

After analysing the questionnaire, the researcher identified a number of trends. Firstly, database administrators working within the ICT industry in Ireland experience of MySQL Community servers is limited. The researcher concluded, one possible reason for this may be due to the fact MySQL Community server does not have the automated tuning capabilities, of competitors. The researcher noted questionnaire participants, were overall quite neutral to the proposed product. The researcher observed this response may be influenced, by the participants experience with similar products they may have used.

The second goal, of the questionnaire research, was to identify a possible feature set for the proposed framework. The researcher found, the participants favoured an online dynamic approach. The participants also desired the proposed solution be capable, of being configured by the database administrator to make indexing and memory recommendations. The researcher then

conducted analysis of this approach. Academic research indicated that this is only a feasible solution, when the database server itself supports self regulation of physical tuning (Schnaitter et al., 2006 and Sockut & Iyer, 2009).

On completion of this analysis, the researcher concluded the framework would implement an offline approach capable of being configured by the database administrator to make indexing and memory recommendations.

**Summary of the experimental pre-test post-test results**

The results of the pre-test post-test analysis provided evidence, that the implemented framework satisfied the hypothesis being tested, in this thesis. The framework is capable of providing recommendations for indexing and system variable selections in MySQL Community Servers.

A limited number of tests (18 in total) were executed during this phase of the project. Appendix J contains the average response time in milliseconds for each test executed during the pre-test phase.

Appendix K contains the recommendations made by the framework at this stage. The framework determined that the System Variable configuration at the time of execution did not require alteration. The framework provided a number of recommendations for index configuration at this stage. The recommendations included unique BTREE indexes, non unique BTREE indexes, recreating index on primary key fields and RTREE unique index.

On analysis the researcher concluded the framework had provided both a realistic and expected recommendation for each test. The researcher then implemented all the recommendations provided by the framework and then commenced post-testing. Appendix L

contains the average response time in milliseconds for each test executed during the post-test phase.

In the post-test phase the response times for all of the 18 tests (Appendix J) improved significantly. The total average response time for all 18 tests combined reduced from 34338ms (Appendix J) to 72ms (Appendix L). No new entries were written to the slow query log file during the post-test phase.

**Project Management**

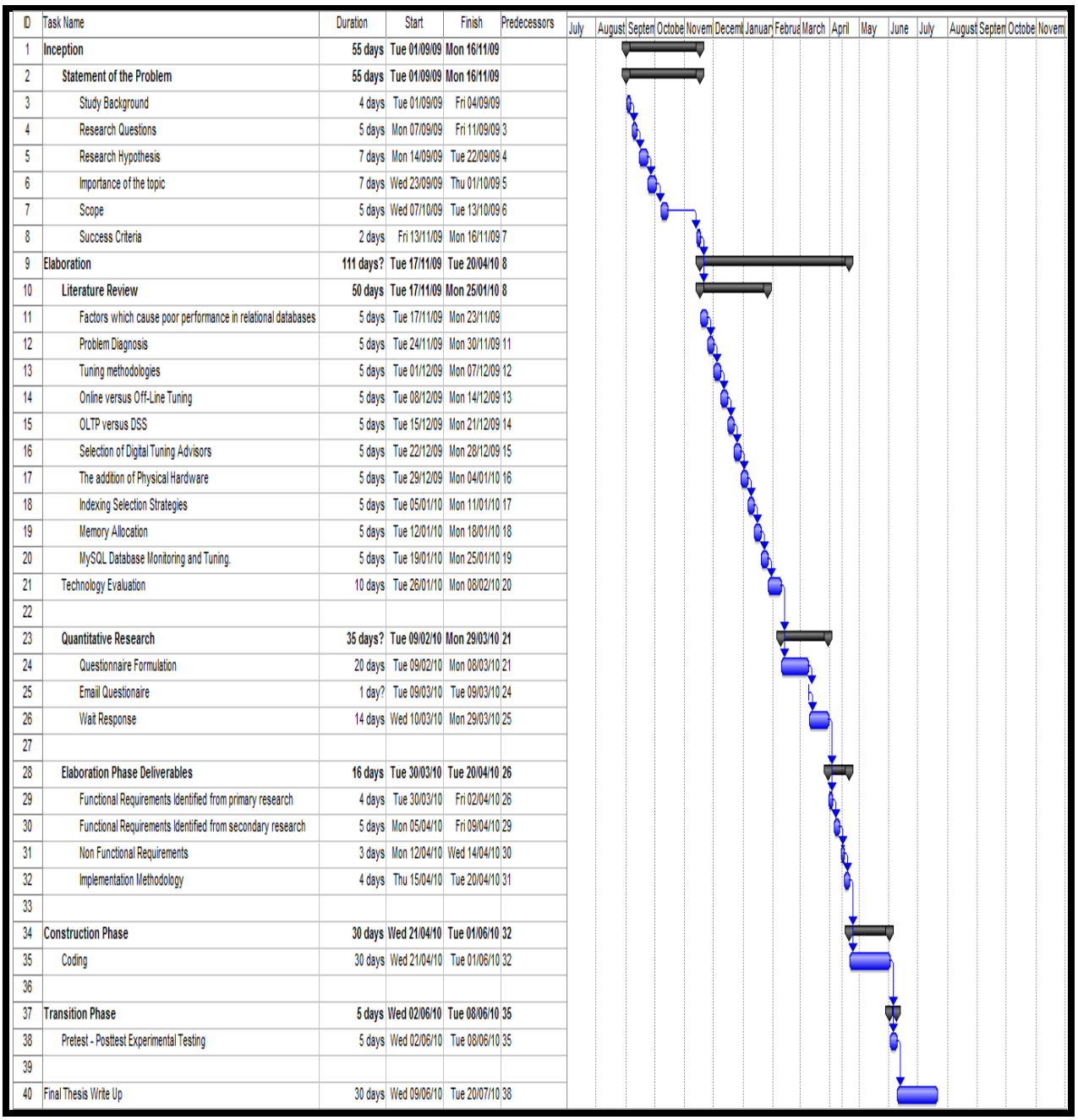
During chapter three, the researcher chose the iterative software development lifecycle, to be used during the course of this project. The researcher followed this process, to manage the project through a full iteration of this lifecycle.

The researcher used the iterative SDLC, to identify the four major milestones for this project. These include inception, elaboration, construction and transition phase. Each milestone required all the relevant key deliverables to be completed, before the project could move to the next phase.

The project commenced with the thesis statement and concluded with the experimental pre-test and post-test results.

**Project Schedule**

Figure 15, contains the project schedule plan used during the course of this project. At the start of the project a schedule was provided by the researcher. Given the resources available the researcher believed the entire project would take approximately 10 months. The project scheduled to commence on the first of September 2009 and to complete on the 20th of July 2010.



| ID | Task Name | Duration | Start | Finish | Predecessors |
|---|---|---|---|---|---|
| 1 | **Inception** | **55 days** | **Tue 01/09/09** | **Mon 16/11/09** | |
| 2 | **Statement of the Problem** | **55 days** | **Tue 01/09/09** | **Mon 16/11/09** | |
| 3 | Study Background | 4 days | Tue 01/09/09 | Fri 04/09/09 | |
| 4 | Research Questions | 5 days | Mon 07/09/09 | Fri 11/09/09 | 3 |
| 5 | Research Hypothesis | 7 days | Mon 14/09/09 | Tue 22/09/09 | 4 |
| 6 | Importance of the topic | 7 days | Wed 23/09/09 | Thu 01/10/09 | 5 |
| 7 | Scope | 5 days | Wed 07/10/09 | Tue 13/10/09 | 6 |
| 8 | Success Criteria | 2 days | Fri 13/11/09 | Mon 16/11/09 | 7 |
| 9 | **Elaboration** | **111 days?** | **Tue 17/11/09** | **Tue 20/04/10** | 8 |
| 10 | **Literature Review** | **50 days** | **Tue 17/11/09** | **Mon 25/01/10** | 8 |
| 11 | Factors which cause poor performance in relational databases | 5 days | Tue 17/11/09 | Mon 23/11/09 | |
| 12 | Problem Diagnosis | 5 days | Tue 24/11/09 | Mon 30/11/09 | 11 |
| 13 | Tuning methodologies | 5 days | Tue 01/12/09 | Mon 07/12/09 | 12 |
| 14 | Online versus Off-Line Tuning | 5 days | Tue 08/12/09 | Mon 14/12/09 | 13 |
| 15 | OLTP versus DSS | 5 days | Tue 15/12/09 | Mon 21/12/09 | 14 |
| 16 | Selection of Digital Tuning Advisors | 5 days | Tue 22/12/09 | Mon 28/12/09 | 15 |
| 17 | The addition of Physical Hardware | 5 days | Tue 29/12/09 | Mon 04/01/10 | 16 |
| 18 | Indexing Selection Strategies | 5 days | Tue 05/01/10 | Mon 11/01/10 | 17 |
| 19 | Memory Allocation | 5 days | Tue 12/01/10 | Mon 18/01/10 | 18 |
| 20 | MySQL Database Monitoring and Tuning. | 5 days | Tue 19/01/10 | Mon 25/01/10 | 19 |
| 21 | Technology Evaluation | 10 days | Tue 26/01/10 | Mon 08/02/10 | 20 |
| 22 | | | | | |
| 23 | **Quantitative Research** | **35 days?** | **Tue 09/02/10** | **Mon 29/03/10** | 21 |
| 24 | Questionnaire Formulation | 20 days | Tue 09/02/10 | Mon 08/03/10 | 21 |
| 25 | Email Questionaire | 1 day? | Tue 09/03/10 | Tue 09/03/10 | 24 |
| 26 | Wait Response | 14 days | Wed 10/03/10 | Mon 29/03/10 | 25 |
| 27 | | | | | |
| 28 | **Elaboration Phase Deliverables** | **16 days** | **Tue 30/03/10** | **Tue 20/04/10** | 26 |
| 29 | Functional Requirements Identified from primary research | 4 days | Tue 30/03/10 | Fri 02/04/10 | 26 |
| 30 | Functional Requirements Identified from secondary research | 5 days | Mon 05/04/10 | Fri 09/04/10 | 29 |
| 31 | Non Functional Requirements | 3 days | Mon 12/04/10 | Wed 14/04/10 | 30 |
| 32 | Implementation Methodology | 4 days | Thu 15/04/10 | Tue 20/04/10 | 31 |
| 33 | | | | | |
| 34 | **Construction Phase** | **30 days** | **Wed 21/04/10** | **Tue 01/06/10** | 32 |
| 35 | Coding | 30 days | Wed 21/04/10 | Tue 01/06/10 | 32 |
| 36 | | | | | |
| 37 | **Transition Phase** | **5 days** | **Wed 02/06/10** | **Tue 08/06/10** | 35 |
| 38 | Pretest - Posttest Experimental Testing | 5 days | Wed 02/06/10 | Tue 08/06/10 | 35 |
| 39 | | | | | |
| 40 | Final Thesis Write Up | 30 days | Wed 09/06/10 | Tue 20/07/10 | 38 |



*Figure 15*. Project Schedule

**Lessons Learnt**

During the course of this project, a number of important lessons were learnt, by the researcher.

- Firstly, the search for the optimal configuration is not of real benefit. A configuration will only be optimal for short periods of time and requires a great deal of effort to be arrived at. Therefore a configuration near optimal, is a much more realistic and achievable goal.
- There are no best practice methods, available towards index and system variable selection for a MySQL Community Server.
- The database administrator should have the controlling influence on whether recommendations made, are implemented. Implementing recommendations, without the prior knowledge of the database administrator often times may be counterproductive.
- Workload is the key factor in formulating recommendations.

**Limitations**

Despite the strengths, a number of issues surrounding this work restrict conclusions, which can be drawn from the results. The researcher acknowledges the following limitations

- Firstly, the approach outlined in this thesis uses an offline analysis and tuning approach. This technique requires regular maintenance windows to be scheduled for physical tuning to occur. This hinders the frameworks ability support databases which must operate in a highly available manner.
- The results of the pre-test post-test research offered as part of ‘proof of concept’ is based on a synthetic workload run under predefined test conditions. Whilst this research clearly proves the tool is successful in this context, further research is required using live database systems.

- As stated during the methodology chapter, this thesis represents the first iteration of the framework. Therefore only simple SQL operations were considered.
- The tool only informs the user if a system resource (key buffer, max connections and so forth) has fallen to a dangerously low value. It does not provide recommendation on what a suitable value for this system variable is.
- Each test was conducted by enabling the slow query log on the MySQL Community server. On databases with high transaction rates this may in itself result in a degradation of performance.
- The framework currently only provides recommendations on index creation. It currently offers no recommendations on the removal of spurious indexes.
- Finally, by making the selection of recommendations configurable the proposed tool is placing a burden on the abilities of the individual database administrator. This can be considered as being of both of benefit and limitation.

**Conclusion**

During this chapter, the researcher discussed some of key findings made during the course of this thesis. The researcher found the JMySQLTune product, had through the experimental pre-test post-test results proved the overall thesis hypothesis. An overview presentation of the project management phase was presented. Finally, the researcher offered some insight to the lessons learnt and limitations identified during the course of the research. The following chapter provides a conclusion of the thesis.

# Chapter 6 - Conclusion

## Introduction

The researcher commenced this project, with the task of developing an automated MySQL Community server tuning framework. The researcher believes, the resulting framework, limitations acknowledged, has proven capable of solving the problems of identifying effective recommendations for index and system variable selection in MySQL community servers.

During this chapter, the researcher discusses the findings, recommendations and conclusions drawn from this study.

## Findings

In the introduction chapter, the researcher identified a number of research questions to be answered, during the course of this thesis. The researcher believed to resolve these questions, a hybrid research methodology was required. Literature review, questionnaire, technology evaluation and experimental pre-test post-test design research were all used during this project. From this research a number of key findings emerged.

During the literature review the researcher established, that the area of performance tuning is one which receives much academic attention with a wide range of approaches being offered. The researcher believed a number of key findings emerged from this review including.

- Database administrators very often suffer from a lack of understanding in relation to these systems and as a result make poor configuration choices, which can lead to major performance issues.
- Many modern databases are now moving towards a self regulation.

- Workload is the key consideration in any database tuning process
- Before tuning commences, database workload must be evaluated, allocating additional system resources to a database, table or query which in large part remains unused is counterproductive.
- Seeking the optimal solution is not a practical approach to tuning. Most current solutions prefer to rely on heuristics.
- A decision tree is an effective and easily understandable approach, for problem resolution.

The researcher from the literature review found, there is currently no 'silver bullet' solution to the tuning problem. All approaches identified claim to improve database performance.

During the technology evaluation the researcher briefly reviewed, a number of tools available to the MySQL database administrator. The researcher concluded, the evaluated tools present a number of common characteristics including

- They have gained support, due to claims of their success in increasing database performance.
- These tools are generally developed by database administrators and MySQL enthusiasts.
- Support for new releases, new functionality and bug fixes are provided at the discretion of these individuals.
- Recommendations issued, are based solely on the capabilities of individual MySQL database enthusiasts. The database administrator executing the tool has no input into how these decisions are made. Instead the DBA's only role is to decide, whether or not to implement the recommendations provided by these scripts.

- No qualitative or quantitative research is available, for these tools. The only evidence of these tools successes, are based on the user feedback posted on internet forums and blogs.
- Little or no user documentation is available, with any of these tools. For all solutions, to determine the logic behind recommendations the database administrator must also be competent in reading high level programming languages.

During the questionnaire research, the researcher wished to establish some high level requirements for the proposed framework. The researcher sought advice on a number of characteristics in regards to this objective.

The researcher believes the key findings, which emerged, from this questionnaire include.

- The participants were overall quite neutral, with regards to using a MySQL community database.
- The participants were more likely to use a commercial database, such as Oracle in their day to day work.
- A consensus emerged amongst the participants, that similar tools did not fully meet expectations with some recording a negative experience.
- This negative experience was largely built on, working with similar tools on other database servers.
- The participants scoring of online dynamic tuning as the most desirable feature reflects the increasing requirement for databases to become available 24*7.
- The least desirable features would be the automatic implementation of recommendations without the approval of the database administrator.

At the end of the elaboration phase, the key findings from the literature review, technology evaluation and questionnaire were evaluated. The researcher used these findings to formulate a full set of functional and non functional requirements for the construction phase. The following popular features were incorporated into the design

- Configurable index selection strategy
- Configurable memory allocation strategy
- Index recommendations for evaluation
- Memory Allocation recommendations for evaluation by the database administrator

The researcher believed the experimental pre-test post-test research, would demonstrate whether the framework satisfied the thesis hypothesis. The findings which emerged during these tests included:

- The recommendations provided by the tool resulted in performance improvements for all tests.
- The tool does not issue recommendations on any select operation which is not present in the slow query log file.
- The tool considers number of distinct rows within tables before issuing indexing recommendations.
- The tool considers number of distinct operations before issuing recommendations
- The tool considers both the type of select operation and the engine before issuing recommendations.
- During the stability phase no performance degradation of any operation in the test scenarios was demonstrable.

**Project Recommendations**

The researcher concluded this study at the end of the transition phase of the SDLC. At this point the researcher believed this thesis, had met its stated aims. However, in future iterations, the researcher believes the framework would benefit from some additional research and development efforts.

At the end of this study, the researcher identified the following functional requirements for future iterations of the JMySQLTune framework

The addition of a Java based graphical user interface (GUI) for the framework with the following functionality

- Support the Database Administrator configure the framework for execution
- Support the Database Administrator configure the Index and System Variable xml files
- A user friendly interface for reporting recommendations

The researcher believes the framework should also be extended with support for more complicated MySQL type operations. In future iterations, the researcher recommends additional support for the following operations

- UNION
- DEPENDENT UNION
- UNION RESULT
- SUBQUERY
- DERIVED
- UNCACHEABLE SUBQUERY
- UNCACHEABLE UNION

During the course of this project, the researcher concentrated on providing index creation recommendations. In future iterations, the framework should provide recommendations for removal of ineffective indexes.

Index and system variable selection are complex tasks, with many variables contributing to the final decision. The researcher believes whilst the pre-test post-test experimental results provide 'proof of concept', they do not represent a full test plan for the framework. Before this software is packaged and delivered to a customer site, the researcher recommends a test strategy should be written. A full suite of tests should be then executed, with all critical and major bugs removed.

**Future Academic Research Recommendations**

The researcher believes the framework, would benefit from additional academic research efforts.

The requirement for systems to be available 24/7, means maintenance windows, are increasingly difficult to schedule. In response to this, online dynamic tuning has recently received much academic attention. Database vendors are attempting to incorporate this approach through self regulation in latest releases.

To receive the benefits of online dynamic tuning, database administrators must upgrade their entire database. This may not always be possible, for a number of reasons including technical, resourcing and pricing.

The researcher believes future investigation, into online dynamic tuning should consider the scenario, where the database is not self regulated. This is a complicated task, fraught with danger as monitoring and implementing erroneous decisions on live databases in an online manner may result in a significant downtime.

**Conclusions**

This thesis commenced with the goal of providing a framework which is capable of tuning a MySQL Community server. During the course of this research, the researcher found the physical tuning of database servers is a complicated task.

This study has assisted the researcher in understanding the issues associated with effectively tuning MySQL community database server. The use of quantitative research, through questionnaire has enabled the researcher to identify the key functionality required by the proposed framework. The evaluation of similar technologies has provided the researcher, with a qualitative insight as to how to best approach the problem. The use of quantitative experimental pre-test post-test research, has demonstrated changes in response times by implementing recommendations made by the framework.

During the course of this thesis, the researcher identified and presented a possible solution to this problem. The thesis outlines the various components, required to detect performance bottlenecks in such system. To achieve this, the tuning process itself is broken into a number of parts, including monitoring, analysis and detection.

From research carried out during the course of the project, the researcher believes one limitation of existing solutions for tuning MySQL Community servers, is the lack of focus on the input of the database administrator. In response to this, the researcher found benefit in using a configurable XML based decision tree approach, for problem analysis and resolution. The framework adopted this approach, for both the indexing and system variable configuration recommendations.

Finally, the researcher outlined a number of test scenarios (Appendix H). These scenarios were then used to evaluate the ability of the framework, to achieve the goal of the

project. The researcher verified the recommendations, complied with the expected results configured in the Index.xml and SystemVariable.xml files. The average response time, for the entire eighteen separate scenarios reduced from 342338 ms (Appendix J) to 72 ms with zero errors (Appendix L).

As planned, the researcher concluded this project at the end of the first iteration of SDLC. The findings of this research, suggest this thesis meets it stated objectives in providing a framework suitable for the automatic physical configuration and tuning of MySQL community servers using scientific processes and repeatable techniques.

## References

Agrawal, R., Ailamaki, A., Bernstein, P. A., Brewer, E. A., Carey, M. J., Chaudhuri, S., el al.,(2008). The Claremont report on database research. *SIGMOD Rec.*, *37*(3), 9-19. doi: 10.1145/1462571.1462573.

Agrawal, S., Chaudhuri, S., Kollar, L., Marathe, A., Narasayya, V., & Syamala, M (2005). Database tuning advisor for microsoft SQL server 2005: demo. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 930-932). Baltimore, Maryland: ACM. doi: 10.1145/1066157.1066292.

Agrawal, S., Chaudhuri, S., & Narasayya, V (2001). Materialized view and index selection tool for Microsoft SQL server 2000. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data* (p. 608). Santa Barbara, California, United States: ACM. doi: 10.1145/375663.375769.

Ahmed, R., Lee, A., Witkowski, A., Das, D., Su, H., Zait, M., el al.,(2006). Cost-based query transformation in Oracle. In *Proceedings of the 32nd international conference on Very large data bases* (pp. 1026-1036). Seoul, Korea: VLDB Endowment. Retrieved July 29, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1182635.1164215&coll=Portal&dl=ACM&CFID=45662376&CFTOKEN=73452892.

Annesley, C (2006). Open source database 'years away from being a serious contender'. *Computer Weekly*, 10. doi: Article.

Bayan, M., & Cangussu, J. W (2008). Automatic feedback, control-based, stress and load testing. In *Proceedings of the 2008 ACM symposium on Applied computing* (pp. 661-666). Fortaleza, Ceara, Brazil: ACM. doi: 10.1145/1363686.1363847.

Benoit, D. G (2005). Automatic Diagnosis of Performance Problems in Database Management Systems. In *Proceedings of the Second International Conference on Automatic Computing* (pp. 326-327). IEEE Computer Society. Retrieved August 13, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1078486&dl=ACM&coll=Portal&CFID=48487895&CFTOKEN=47747821

Belani, H., Car, Z., & Carić, A (2009). RUP-Based process model for security requirements engineering in value-added service development. In *Proceedings of the 5th International Workshop on Software Engineering for Secure Systems, Vancouver, Canada* (pp. 54–60).

Biskup, J (1983). A foundation of CODD's relational maybe-operations. *ACM Trans. Database Syst.*, *8*(4), 608-636. doi: 10.1145/319996.320014.

Boehm, B., & Hansen, W (2001). The spiral model as a tool for evolutionary acquisition. *CrossTalk*, *14*(5), 4–11.

Bruno, N., & Chaudhuri, S (2005). Automatic physical database tuning: a relaxation-based approach. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 227-238). Baltimore, Maryland: ACM. doi: 10.1145/1066157.1066184.

Bruno, N., & Chaudhuri, S (2006). To tune or not to tune?: a lightweight physical design alerter. In *Proceedings of the 32nd international conference on Very large data bases* (pp. 499-510). Seoul, Korea: VLDB Endowment. Retrieved July 24, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1182635.1164171&coll=Portal&dl=ACM&CFID=46443895&CFTOKEN=57659103.

Bruno, N., & Chaudhuri, S (2007a). Online autoadmin: (physical design tuning). In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 1067-1069). Beijing, China: ACM. doi: 10.1145/1247480.1247608.

Bruno, N., & Chaudhuri, S (2007b). Physical design refinement: The \‘merge-reduce\’ approach. *ACM Trans. Database Syst., 32*(4), 28. doi: 10.1145/1292609.1292618.

Bujdei, C., Moraru, S., & Dan, S (2008). Optimize databases for health monitoring systems. In *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments* (pp. 1-8). Athens, Greece: ACM. doi: 10.1145/1389586.1389676.

Burd, B (2006). *Java for dummies*. For Dummies.

Cabral, S. K., & Keith Murphy ( (2009). *MySQL Administrator's Bible* (p. 888). Wiley Publishing. Retrieved January 7, 2010, from http://portal.acm.org/citation.cfm?id=1643594.

Cardelli, L., & Wegner, P (1985). On understanding types, data abstraction and polymorphism. *ACM Comput. Surv., 17*(4), 471-523. doi: 10.1145/6041.6042.

Chaudhuri, S., & Narasayya, V (1998). AutoAdmin \“what-if\” index analysis utility. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (pp. 367-378). Seattle, Washington, United States: ACM. doi: 10.1145/276304.276337.

Chaudhuri, S., & Narasayya, V (2007). Self-tuning database systems: a decade of progress. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 3-14). Vienna, Austria: VLDB Endowment. Retrieved July 29, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1325851.1325856&coll=Portal&dl=ACM&CFID=45688813&CFTOKEN=83576216.

Chaudhuri, S., & Narasayya, V. R (1997). An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the 23rd International Conference on Very Large Data Bases*

(pp. 146-155). Morgan Kaufmann Publishers Inc. Retrieved September 23, 2009, from http://www.vldb.org/conf/1997/P146.PDF.

Chaudhuri, S., & Weikum, G (2005). Foundations of automated database tuning. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 964-965). Baltimore, Maryland: ACM. doi: 10.1145/1066157.1066305.

Chen, Z., Gehrke, J., & Korn, F (2001). Query optimization in compressed database systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data* (pp. 271-282). Santa Barbara, California, United States: ACM. doi: 10.1145/375663.375692.

Chung, I., & Hollingsworth, J. K (2004). Using Information from Prior Runs to Improve Automated Tuning Systems. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing* (p. 30). IEEE Computer Society. Retrieved August 20, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1048933.1049974&coll=Portal&dl=ACM&CFID=48157413&CFTOKEN=58734191.

Comer, D (1978). The difficulty of optimum index selection. *ACM Trans. Database Syst., 3*(4), 440-445. doi: 10.1145/320289.320296.

Dageville, B., Das, D., Dias, K., Yagoub, K., Zait, M., & Ziauddin, M (2004). Automatic SQL tuning in oracle 10g. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* (pp. 1098-1109). Toronto, Canada: VLDB Endowment. Retrieved July 31, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1316689.1316784&coll=Portal&dl=GUIDE&CFID=47099211&CFTOKEN=59625626.

Dageville, B., & Zait, M (2002a). SQL memory management in Oracle9i. In *Proceedings of the 28th international conference on Very Large Data Bases* (pp. 962-973). Hong Kong, China: VLDB

Endowment. Retrieved August 10, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1287454&dl=ACM&coll=Portal&CFID=48190571&CFTOKEN=72295952.

Dageville, B., & Zait, M (2002b). SQL memory management in Oracle9i. In *Proceedings of the 28th international conference on Very Large Data Bases* (pp. 962-973). Hong Kong, China: VLDB Endowment. Retrieved December 17, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1287369.1287454&coll=Portal&dl=ACM&CFID=67117157&CFTOKEN=68709367.

Davis, M (2003). Theoretical foundations for experiential systems design. In *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence* (pp. 45-52). Berkeley, California: ACM. doi: 10.1145/982484.982491.

Deitel, P., & Deitel, H (2006). Java™ how to program.

Di Giacomo M (2005, May 1). MySQL: Lessons Learned on a Digital Library. text, . Retrieved July 28, 2009, from http://www2.computer.org/portal/web/csdl/doi/10.1109/MS.2005.71.

Elnaffar, S. S (2002). A methodology for auto-recognizing DBMS workloads. In *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research* (p. 2). Toronto, Ontario, Canada: IBM Press. Retrieved August 20, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=782115.782117&coll=Portal&dl=ACM&CFID=48115736&CFTOKEN=15655660.

Eriksson, H. E., Penker, M., Lyons, B., & Fado, D (2004). *UML 2 toolkit*. Wiley.

Finkelstein, S., Schkolnick, M., & Tiberio, P (1988). Physical database design for relational databases. *ACM Trans. Database Syst., 13*(1), 91-128. doi: 10.1145/42201.42205.

Foundation, E (n.d.). Eclipse Project. *Web pages at http://www. eclipse. org*.

Gorea, D., & Buraga, S (2006). Towards integrating decision tree with xml technologies. In *Proceedings of the 8th International Conference on Development and Application Systems–DAS*.

Gray, J (2004). The next database revolution. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (pp. 1-4). Paris, France: ACM. doi: 10.1145/1007568.1007570.

Hammer, M., & Chan, A (1976). Index selection in a self-adaptive data base management system. In *Proceedings of the 1976 ACM SIGMOD international conference on Management of data* (pp. 1-8). Washington, D.C.: ACM. Retrieved August 25, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=509383.509385&coll=Portal&dl=ACM&CFID=48811291&CFTOKEN=68747452

Hayden, M (2010). MySQLTuner . *Searching for a new MySQLTuner maintainer*. Retrieved from MySQLTuner.pl Retrieved February 03, 2010, from http://blog.mysqltuner.com/

He, Z., Lee, B. S., & Snapp, R (2005). Self-tuning cost modeling of user-defined functions in an object-relational DBMS. *ACM Trans. Database Syst., 30*(3), 812-853. doi: 10.1145/1093382.1093387.

Henderson, M., Cutt, B., & Lawrence, R (2009). Exploiting join cardinality for faster hash joins. In *Proceedings of the 2009 ACM symposium on Applied Computing* (pp. 1549-1554). Honolulu, Hawaii: ACM. doi: 10.1145/1529282.1529629.

Hu, Y., Sundara, S., & Srinivasan, J (2007). Supporting time-constrained SQL queries in oracle. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 1207-1218).

Vienna, Austria: VLDB Endowment. Retrieved July 29, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1325851.1325989&coll=Portal&dl=ACM&CFID=45662376&CFTOKEN=73452892

Ip, M., Saxton, L., & Raghavan, V (1983). On the Selection of an Optimal Set of Indexes. *Software Engineering, IEEE Transactions on*, *SE-9*(2), 135-143.

Ko, B., Lee, K., Amiri, K., & Calo, S (2003). Scalable Service Differentiation in a Shared Storage Cache. In *Proceedings of the 23rd International Conference on Distributed Computing Systems* (p. 184). IEEE Computer Society. Retrieved January 5, 2010, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=850929.851963&coll=Portal&dl=GUIDE&CFID=69772859&CFTOKEN=67193021

Kofler, M., & Kramer, D (2005). *The definitive guide to MySQL* (p. 785). Apress.

Konchady, M (1998). An Introduction to JDBC. *Linux J.*, *1998*(55es), 2

Kormilitsin, M., Chirkova, R., Fathi, Y., & Stallmann, M (2008). View and index selection for query-performance improvement: quality-centered algorithms and heuristics. In *Proceeding of the 17th ACM conference on Information and knowledge management* (pp. 1329-1330). Napa Valley, California, USA: ACM. doi: 10.1145/1458082.1458261.

Kroll, P (2001). The Spirit of the RUP. *Rational Software*.

Linda Dailey Paulson (2004, July 1). Open Source Databases Move into the Marketplace. text, . Retrieved September 2, 2009, from http://www2.computer.org/portal/web/csdl/doi/10.1109/MC.2004.62

Larman, C., & Basili, V. R (2003). Iterative and incremental development: A brief history. *Computer*, 47–56.

Leedy, P., Ormrod J (2005). *Practical research - planning and design*. Upper saddle River, NJ: Pearson education international.

Lum, V. Y (1974). On the selection of secondary indexes. In *Proceedings of the 1974 annual ACM conference - Volume 2* (pp. 736-736). ACM. doi: 10.1145/1408800.1408901.

Lungu, I., & Vatuiu, T (2008). Manageability comparison: Oracle database 10g and Oracle 9i database. *VOL. VIII PART I*, *8*(1), 295–300.

Manegold, S., Boncz, P. A., & Kersten, M. L (2000). Optimizing database architecture for the new bottleneck: memory access. *The VLDB Journal*, *9*(3), 231-246.

Mission—Speed, M (n.d.). Introduction to MySQL.

Moussa, M., Ruwanpura, J. Y., & Jergeas, G (2004). Decision tree module within decision support simulation system. In *Proceedings of the 36th conference on Winter simulation* (pp. 1268-1276). Washington, D.C.: Winter Simulation Conference. Retrieved September 15, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1161734.1161967&coll=Portal&dl=GUIDE&CFID=52939072&CFTOKEN=73692752

MySQL :: MySQL 6.0 Reference Manual (n.d.). . Retrieved August 7, 2009, from http://dev.mysql.com/doc/refman/6.0/en/index.html

MySQL :: MySQL Enterprise Features (n.d.). . Retrieved November 1, 2009, from http://www.mysql.com/products/enterprise/features.html

Nash, D (1978). Topics in design automation data bases. In *Proceedings of the 15th conference on Design automation* (pp. 463-474). Las Vegas, Nevada, United States: IEEE Press. Retrieved May 21, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=800095.803130&coll=Portal&dl=GUIDE&CFID=36660375&CFTOKEN=82805649

Navathe, S. B (1992). Evolution of data modeling for databases. *Commun. ACM*, *35*(9), 112-123. doi: 10.1145/130994.131001.

Oaks, S., & Wong, H (2004). *Java threads* (p. 340). O'Reilly Media, Inc.

Offermann, P., Levina, O., Schönherr, M., & Bub, U. (2009). Outline of a design science research process. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology* (pp. 1-11). Philadelphia, Pennsylvania: ACM. doi: 10.1145/1555619.1555629

Orlikowski, W. J., & Gash, D. C (1994). Technological frames: making sense of information technology in organizations. *ACM Trans. Inf. Syst.*, *12*(2), 174-207. doi: 10.1145/196734.196745.

Osogami, T., & Kato, S (2007). Optimizing system configurations quickly by guessing at the performance. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (pp. 145-156). San Diego, California, USA: ACM. doi: 10.1145/1254882.1254899.

ParaTools, I., & Eugene, O. R (n.d.). Making Performance Analysis and Tuning Part of the Software Development Cycle.

Powell, G., & McCullough-Dieter, C (2006). *Oracle 10g Database Administrator* (p. 744). Course Technology.

Pro MySQL - Books24x7 (n.d.). . Retrieved July 28, 2009, from http://library.books24x7.com.dml.regis.edu/book/id_12607/viewer.asp?bookid=12607&rowid=141

Qiao, L., Soetarman, B., Fuh, G., Pannu, A., Cui, B., Beavin, T., el al.,(2007). A framework for enforcing application policies in database systems. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 981-992). Beijing, China: ACM. doi: 10.1145/1247480.1247597.

Schnaitter, K., Abiteboul, S., Milo, T., & Polyzotis, N (2006). COLT: continuous on-line tuning. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (pp. 793-795). Chicago, IL, USA: ACM. doi: 10.1145/1142473.1142592.

Schneller, E. V. D (2009, Mittwoch, Dezember 2). Index Analyzer for MySQL - r123. *Index Analyzer for MySQL*. Retrieved February 3, 2010, from http://mysql-index-analyzer.blogspot.com/2009/12/index-analyzer-for-mysql-r123.html.

Sharma, V (2009, June 23). Cost Based Optimizer: Inefficient Input yields Inefficient Output. *Database / SQL Experiences*. Retrieved July 24, 2009, from http://viveklsharma.blogspot.com/2009/06/cost-based-optimizer-inefficient-input.html

Shasha, D (1996). Tuning databases for high performance. *ACM Comput. Surv., 28*(1), 113-115. doi: 10.1145/234313.234363.

Slivinskas, G., Jensen, C. S., & Snodgrass, R. T (2001). Adaptable query optimization and evaluation in temporal middleware. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data* (pp. 127-138). Santa Barbara, California, United States: ACM. doi: 10.1145/375663.375678.

Sockut, G. H., & Iyer, B. R (2009). Online reorganization of databases. *ACM Comput. Surv., 41*(3), 1-136. doi: 10.1145/1541880.1541881.

Stancu-Mara, S., & Baumann, P (2008). A comparative benchmark of large objects in relational databases. In *Proceedings of the 2008 international symposium on Database engineering & applications* (pp. 277-284). Coimbra, Portugal: ACM. doi: 10.1145/1451940.1451980.

Stephens, J., & Russell, C (2004). *Beginning MySQL database design and optimization*. Apress.

Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., Diao, Y., & Surendra, M (2006). Adaptive self-tuning memory in DB2. In *Proceedings of the 32nd international conference on Very large data bases* (pp. 1081-1092). Seoul, Korea: VLDB Endowment. Retrieved August 27, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1182635.1164220&coll=Portal&dl=ACM&CFID=50224897&CFTOKEN=94379955

Stress testing real-time systems with genetic algorithms (n.d.). . Retrieved May 27, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1068009.1068183&jmp=cit&coll=Portal&dl=ACM&CFID=37666108&CFTOKEN=40235388#CIT

Stress testing real-time systems with genetic algorithms (n.d.). . Retrieved May 27, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1068009.1068183&jmp=cit&coll=Portal&dl=ACM&CFID=37666108&CFTOKEN=40235388#CIT

Talebi, Z. A., Chirkova, R., Fathi, Y., & Stallmann, M (2008). Exact and inexact methods for selecting views and indexes for OLAP performance improvement. In *Proceedings of the 11th*

*international conference on Extending database technology: Advances in database technology* (pp. 311-322). Nantes, France: ACM. doi: 10.1145/1353343.1353383.

Tran, D. N., Huynh, P. C., Tay, Y. C., & Tung, A. K. H (2008). A new approach to dynamic self-tuning of database buffers. *Trans. Storage*, *4*(1), 1-25. doi: 10.1145/1353452.1353455.

Montgomory M (2009) tuning-primer.sh. Retrieved July 29, 2009, from http://www.day32.com/MySQL/tuning-primer.sh

Vaishnavi, V., & Kuechler, W. (n.d.). Design Research in Information Systems. Retrieved May 24, 2010, from http://desrist.org/design-research-in-information-systems/

Vuduc, R., Demmel, J. W., & Bilmes, J. A (2004). Statistical Models for Empirical Search-Based Performance Tuning. *International Journal of High Performance Computing Applications*, *18*(1), 65-94. doi: 10.1177/1094342004041293.

Weikum, G., Moenkeberg, A., Hasse, C., & Zabback, P (2002). Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proceedings of the 28th international conference on Very Large Data Bases* (pp. 20-31). Hong Kong, China: VLDB Endowment. Retrieved August 17, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1287369.1287373&coll=GUIDE&dl=GUIDE&CFID=48946147&CFTOKEN=60562462

Wiese, D., Rabinovitch, G., Reichert, M., & Arenswald, S (2008). Autonomic tuning expert: a framework for best-practice oriented autonomic database tuning. In *Proceedings of the 2008*

*conference of the center for advanced studies on collaborative research: meeting of minds* (pp. 27-41). Ontario, Canada: ACM. doi: 10.1145/1463788.1463792.

Xu, J (2008). Rule-based automatic software performance diagnosis and improvement. In *Proceedings of the 7th international workshop on Software and performance* (pp. 1-12). Princeton, NJ, USA: ACM. doi: 10.1145/1383559.1383561.

Young, E (n.d.). Database Monitoring, a Transaction Processing Approach.

Zaman, M., Surabattula, J., & Gruenwald, L (2004). An auto-indexing technique for databases based on clustering. In *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on* (pp. 776-780). Presented at the Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on.

Zawodny, J. D., & Balling, D. J (2004). *High Performance MySQL* (p. 276). O'Reilly.

Yin, R. K (2003). *Applications of case study research*. Sage Publications, Inc.

Zenmyo, T (2009). Identifying performance bottlenecks based on the local parameter tuning. In *Proceedings of the 6th international conference on Autonomic computing* (pp. 53-54). Barcelona, Spain: ACM. doi: 10.1145/1555228.1555242.

# APPENDICES

## Appendix A: Questionnaire

### 1. Reasons for this study

My name is Kevin Spillane; I am currently completing my final year thesis for a Masters Degree in Software Information Systems offered by NUI Galway.

The purpose of this study is to gather valuable information, which will assist this thesis in determining the requirements of an automated database tuning framework for MySQL Community servers.

### 2. Your involvement in this research study

I would like to invite you to take part in this important survey research due to your experience of tuning and maintaining relational databases as part of your role.

If you agree to participate in this study please complete the attached electronic survey questionnaire. The information that I need to gather from you includes: the most common issues associated with manually maintaining.

When you have completed this e-mailed questionnaire please send it back to me electronically **at kevis23@hotmail.com.**

If you wish to receive a summary of the results (that you can pass on to your home company) please indicate at the end of this questionnaire and include your e-mail address.

### 3. Issues of confidentiality

Any information that you provide will be confidential to the researchers. All participants will be anonymous such that no personal information concerning you or your company will be made public either during, or after the completion and release of this study. During this study, no-one else will have access to any participants answered questionnaires. The questionnaires will be destroyed once the study has been completed.

**Q1. How interested would you be in using JMySQLTune, an automated tuning product which improves performance on an open-source MySQL community server?**

[ ] Very Interested

[ ] Interested

[ ] Neutral

[ ] Uninterested

[ ] Very Uninterested

**Q2. How many years experience have you gained working with the following vendors databases?**

| | **0 months** | **< 6 months** | **< 1 year** | **1 – 5 years** | **> 5 years** |
|---|---|---|---|---|---|
| **Oracle** | [ ] | [ ] | [ ] | [ ] | [ ] |
| **Sybase** | [ ] | [ ] | [ ] | [ ] | [ ] |
| **MySQL** | [ ] | [ ] | [ ] | [ ] | [ ] |
| **Other** | [ ] | [ ] | [ ] | [ ] | [ ] |

**Q3. With regards the following features, Please give a ranking from 1 – 8.**
**(1 being the most important and 8 least important).**

| **Feature** | **Ranking** |
|---|---|
| Automatic updating (insert/removal) indexes | _____ |
| Automatic updating of memory configuration variables | _____ |
| Configurable index selection strategy | _____ |
| Configurable memory allocation strategy | _____ |
| Graphical User Interface | _____ |
| Index recommendations for evaluation | _____ |
| Memory Allocation recommendation s for evaluation | _____ |
| Online dynamic tuning | _____ |

**Q4. How likely are you to use an automated tuning advisor to obtain index recommendations for a database server?**

[ ] Very Interested

[ ] Interested

[ ] Neutral

[ ] Uninterested

[ ] Very Uninterested

**Q5. How likely are you to use an automated tuning advisor to obtain cache recommendations for a database server?**

[ ] Very Interested

[ ] Interested

[ ] Neutral

[ ] Uninterested

[ ] Very Uninterested

**Q6. If this product were available today, would use it for tuning MySQL Community servers?**

[ ] Very Interested

[ ] Interested

[ ] Neutral

[ ] Uninterested

[ ] Very Uninterested

**Q7. Are there additional features, you feel should incorporated into this product?**

**Q8. Please indicate the degree to which you agree/disagree with the following statements about your experience of automated database tuning products**

| | Agree | Neutral | Disagree |
|---|---|---|---|
| They mostly improved performance of indexes | [ ] | [ ] | [ ] |
| They mostly improved performance of memory | [ ] | [ ] | [ ] |
| They were configurable | [ ] | [ ] | [ ] |
| They improved the overall system performance | [ ] | [ ] | [ ] |
| They were user friendly and easy to use | [ ] | [ ] | [ ] |

**Q9. How would you currently tune indexes and/or cache memory in a MySQL community server?**

[ ] I don't know I have never tried

[ ] Using a Perl script I downloaded from the internet

[ ] Using manual intervention

[ ] Other

**Q10. Please provide additional comments you may have**

_____

_____

_____

_____

**Q11. Would you like to receive feedback with regards to the results of this survey?**

[ ] No

[ ] Yes, please email results to _____

**Appendix B: SystemVariable.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<tree>
 <branch id="ROOT">
  <decision>percentage</decision>
  <value>85</value>
  <yes>thread_cache_hit_rate</yes>
  <no>thread_cache_hit</no>
  <message>Threads_created/Connections</message>
 </branch>
 <branch id="thread_cache_hit_rate">
  <decision>display</decision>
  <value>thread cache hit rate greater than 85% of connections</value>
  <yes>aborted_connections</yes>
  <no>aborted_connections</no>
 </branch>
 <branch id="thread_cache_hit">
  <decision>display</decision>
  <value>thread cache hit rate ok</value>
  <yes>aborted_connections</yes>
  <no>aborted_connections</no>
 </branch>
 <branch id="aborted_connections">
  <decision>percentage</decision>
  <value>80</value>
  <yes>aborted_connection_warning</yes>
  <no>aborted_connection_ok</no>
  <message>Aborted_connects/Connections</message>
 </branch>
 <branch id="aborted_connection_warning">
  <decision>display</decision>
  <value>aborted connections greater than 85% of connections</value>
  <yes>table_locks_immediate</yes>
  <no>table_locks_immediate</no>
 </branch>
 <branch id="aborted_connection_ok">
  <decision>display</decision>
  <value>aborted connections ok</value>
  <yes>table_locks_immediate</yes>
  <no>table_locks_immediate</no>
 </branch>
 <branch id="table_locks_immediate">
  <decision>percentage</decision>
  <value>90</value>
  <yes>table_locks_ok</yes>
  <no>table_locks_warning</no>
```

```
  <message>Table_locks_immediate/Table_locks_waited+Table_locks_immediate</message>
</branch>
<branch id="table_locks_warning">
  <decision>display</decision>
  <value>Table_locks_immediate less than 90% of locks</value>
  <yes>key_cache</yes>
  <no>key_cache</no>
</branch>
<branch id="table_locks_ok">
  <decision>display</decision>
  <value>Table_locks_immediate ok</value>
  <yes>query_cache</yes>
  <no>query_cache</no>
</branch>
<branch id="query_cache">
  <decision>percentage</decision>
  <value>10</value>
  <yes>query_cache_ok</yes>
  <no>query_cache_warning</no>
  <message>Qcache_hits/Com_select+Qcache_hits</message>
</branch>
<branch id="query_cache_warning">
  <decision>display</decision>
  <value>Qcache_hits less than 10% of selects</value>
  <yes>key_cache</yes>
  <no>key_cache</no>
</branch>
<branch id="query_cache_ok">
  <decision>display</decision>
  <value>Qcache_hits ok</value>
  <yes>key_cache</yes>
  <no>key_cache</no>
</branch>
<branch id="key_cache">
  <decision>percentage</decision>
  <value>10</value>
  <yes>key_cache_error</yes>
  <no>key_cache_ok</no>
  <message>Key_reads/Key_read_requests</message>
</branch>
<branch id="key_cache_error">
  <decision>display</decision>
  <value>Key_reads less than 10% of Key_read_requests</value>
  <yes>key_buffers</yes>
  <no>key_buffers</no>
</branch>
```

```
<branch id="key_cache_ok">
  <decision>display</decision>
  <value>key cache ok</value>
  <yes>key_buffers</yes>
  <no>key_buffers</no>
</branch>
<branch id="key_buffers">
  <decision>percentage</decision>
  <value>85</value>
  <yes>key_buffers_error</yes>
  <no>key_buffers_ok</no>
  <message>key_cache_block_size/key_buffer_size</message>
</branch>
<branch id="key_buffers_error">
  <decision>display</decision>
  <value>key_cache_block_size greater than 85% of key_buffer_size</value>
  <yes>slow_queries</yes>
  <no>slow_queries</no>
</branch>
<branch id="key_buffers_ok">
  <decision>display</decision>
  <value>key_cache_block_size ok</value>
  <yes>slow_queries</yes>
  <no>slow_queries</no>
</branch>
<branch id="slow_queries">
  <decision>percentage</decision>
  <value>10</value>
  <yes>slow_queries_error</yes>
  <no>slow_queries_ok</no>
  <message>Slow_queries/Questions</message>
</branch>
<branch id="slow_queries_error">
  <decision>display</decision>
  <value>More than 10% of queries are slow</value>
  <yes>max_used_connection</yes>
  <no>max_used_connection</no>
</branch>
<branch id="slow_queries_ok">
  <decision>display</decision>
  <value>Slow Queries ok</value>
  <yes>max_used_connection</yes>
  <no>max_used_connection</no>
</branch>
<branch id="max_used_connection">
  <decision>percentage</decision>
```

```
  <value>90</value>
  <yes>max_used_connections_error</yes>
  <no>max_used_connections_ok</no>
  <message>Max_used_connections/max_connections</message>
 </branch>
 <branch id="max_used_connections_ok">
  <decision>display</decision>
  <value>Max_used_connections Ok</value>
  <yes>pct_temp_sort_table</yes>
  <no>pct_temp_sort_table</no>
 </branch>
 <branch id="max_used_connections_error">
  <decision>display</decision>
  <value>More than 90% of Connections used</value>
  <yes>pct_temp_sort_table</yes>
  <no>pct_temp_sort_table</no>
 </branch>
 <branch id="pct_temp_sort_table">
  <decision>percentage</decision>
  <value>90</value>
  <yes>pct_temp_sort_table_error</yes>
  <no>pct_temp_sort_table_ok</no>
  <message>Sort_merge_passes/Sort_scan+Sort_range</message>
 </branch>
 <branch id="pct_temp_sort_table_ok">
  <decision>display</decision>
  <value>Temporary table sorts Ok</value>
  <yes>slow_queries</yes>
  <no>slow_queries</no>
 </branch>
 <branch id="pct_temp_sort_table_error">
  <decision>display</decision>
  <value>More than 90% of sorts used temporary tables</value>
  <yes>pct_temp_sort_table</yes>
  <no>pct_temp_sort_table</no>
 </branch>
</tree>
```

**Appendix C Index.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<tree>
  <branch id="ROOT">
    <decision>select_type</decision>
    <value>SIMPLE</value>
    <yes>POSSIBLE_KEYS</yes>
    <no>PRIMARY</no>
  </branch>
  <branch id="PRIMARY">
    <decision>select_type</decision>
    <value>PRIMARY</value>
    <yes>RECREATE_PRIMARY_KEY_INDEX</yes>
    <no>MUL</no>
  </branch>
  <branch id="UNION">
    <decision>select_type</decision>
    <value>UNION</value>
    <yes>UNSUPPORTED OPERATION : UNION</yes>
    <no>NULL</no>
  </branch>
  <branch id="DEPENDENT_UNION">
    <decision>display</decision>
    <value>DEPENDENT UNION</value>
    <yes>UNSUPPORTED OPERATION : DEPENDENT UNION</yes>
    <no>UNSUPPORTED OPERATION : DEPENDENT UNION</no>
  </branch>
  <branch id="UNION_RESULT">
    <decision>display</decision>
    <value>UNION RESULT</value>
    <yes>UNSUPPORTED OPERATION : UNION RESULT</yes>
    <no>UNSUPPORTED OPERATION : UNION RESULTS</no>
  </branch>
  <branch id="SUBQUERY">
    <decision>display</decision>
    <value>UNION RESULT</value>
    <yes>UNSUPPORTED OPERATION : SUBQUERY</yes>
    <no>UNSUPPORTED OPERATION : SUBQUERY</no>
  </branch>
  <branch id="DEPENDENT_SUBQUERY">
    <decision>display</decision>
    <value>UNION RESULT</value>
    <yes>UNSUPPORTED OPERATION : DEPENDENT SUBQUERY</yes>
```

```
  <no>UNSUPPORTED OPERATION : DEPENDENT SUBQUERY</no>
 </branch>
 <branch id="DERIVED">
  <decision>display</decision>
  <value>UNION RESULT</value>
  <yes>UNSUPPORTED OPERATION : DERIVED</yes>
  <no>UNSUPPORTED OPERATION : DERIVED</no>
 </branch>
 <branch id="UNCACHEABLE_SUBQUERY">
  <decision>display</decision>
  <value>UNION RESULT</value>
  <yes>UNSUPPORTED OPERATION : UNCACHEABLE SUBQUERY</yes>
  <no>UNSUPPORTED OPERATION : UNCACHEABLE SUBQUERY</no>
 </branch>
 <branch id="UNCACHEABLE_UNION">
  <decision>display</decision>
  <value>UNION RESULT</value>
  <yes>UNSUPPORTED OPERATION : UNCACHEABLE UNION</yes>
  <no>UNSUPPORTED OPERATION : UNCACHEABLE UNION</no>
 </branch>
 <branch id="POSSIBLE_KEYS">
  <decision>possible_keys</decision>
  <value>null</value>
  <yes>KEY</yes>
  <no>ROWS</no>
 </branch>
 <branch id="KEY">
  <decision>key</decision>
  <value>null</value>
  <yes>DISTINCT_VALUES</yes>
  <no>KEYS EXIST FOR THIS QUERY, INVESTIGATE REREATING INDEX</no>
 </branch>
 <branch id="ROWS">
  <decision>rows</decision>
  <value>10000</value>
  <yes>DISTINCT_VALUES</yes>
  <no>ROWS</no>
 </branch>
 <branch id="DISTINCT_VALUES">
  <decision>distinct</decision>
  <value>1.0000</value>
  <yes>UNIQUE_INDEX_InnoDB</yes>
  <no>NON_UNIQUE_INDEX_InnoDB</no>
 </branch>
 <branch id="NON_DISTINCT_VALUES">
  <decision>distinct</decision>
```

```
  <value>.9000</value>
  <yes>NON_UNIQUE_INDEX_InnoDB</yes>
  <no>NO_ACTION</no>
 </branch>
 <branch id="UNIQUE_INDEX_InnoDB">
  <decision>engine</decision>
  <value>InnoDB</value>
  <yes>BTREE_UNIQUE_INDEX</yes>
  <no>UNIQUE_INDEX_MyISAM</no>
 </branch>
 <branch id="UNIQUE_INDEX_MyISAM">
  <decision>engine</decision>
  <value>MyISAM</value>
  <yes>RANGE_QUERY_UNIQUE_INDEX</yes>
  <no>UNIQUE_INDEX_Hash</no>
 </branch>
 <branch id="NON_UNIQUE_INDEX_InnoDB">
  <decision>engine</decision>
  <value>InnoDB</value>
  <yes>BTREE_NON_UNIQUE_INDEX</yes>
  <no>NON_UNIQUE_INDEX_MyISAM</no>
 </branch>
 <branch id="NON_UNIQUE_INDEX_MyISAM">
  <decision>engine</decision>
  <value>MyISAM</value>
  <yes>RANGE_QUERY_NON_UNIQUE_INDEX</yes>
  <no>NON_UNIQUE_INDEX_Hash</no>
 </branch>
 <branch id="RANGE_QUERY_UNIQUE_INDEX">
  <decision>engine</decision>
  <value>null</value>
  <yes>BTREE_UNIQUE_INDEX</yes>
  <no>RTREE_UNIQUE_INDEX</no>
 </branch>
 <branch id="RANGE_QUERY_NON_UNIQUE_INDEX">
  <decision>engine</decision>
  <value>null</value>
  <yes>BTREE_NON_UNIQUE_INDEX</yes>
  <no>RTREE_NON_UNIQUE_INDEX</no>
 </branch>
 <branch id="BTREE_UNIQUE_INDEX">
  <decision>display</decision>
  <value>InnoDB</value>
  <yes>BTREE UNIQUE INDEX</yes>
  <no>ROOT</no>
 </branch>
```

```
  <branch id="BTREE_NON_UNIQUE_INDEX">
    <decision>display</decision>
    <value>InnoDB</value>
    <yes>BTREE NON UNIQUE INDEX</yes>
    <no>ROOT</no>
  </branch>
  <branch id="RTREE_UNIQUE_INDEX">
    <decision>display</decision>
    <value>InnoDB</value>
    <yes>BTREE UNIQUE INDEX</yes>
    <no>ROOT</no>
  </branch>
  <branch id="RTREE_NON_UNIQUE_INDEX">
    <decision>display</decision>
    <value>InnoDB</value>
    <yes>BTREE NON UNIQUE INDEX</yes>
    <no>ROOT</no>
  </branch>
  <branch id="NO_ACTION">
    <decision>display</decision>
    <value>InnoDB</value>
    <yes>NO ACTION</yes>
    <no>ROOT</no>
  </branch>
</tree>
```

**Appendix D: show status**

```
mysql> show status;

+----------------------------------+-----------+

| Variable_name                    | Value     |

+----------------------------------+-----------+

| Aborted_clients                  | 0         |

| Aborted_connects                 | 2         |

| Binlog_cache_disk_use            | 0         |

| Binlog_cache_use                 | 0         |

| Bytes_received                   | 855       |

| Bytes_sent                       | 14552     |

| Com_change_db                    | 0         |


..

..


 Threads_cached                   | 0         |

| Threads_connected                | 1         |

| Threads_created                  | 1         |

| Threads_running                  | 1         |

| Uptime                           | 5366      |

| Uptime_since_flush_status        | 5366      |

+----------------------------------+-----------+

249 rows in set (0.00 sec)


mysql>
```

**Appendix E: Explain command**

```
mysql> explain Table_1;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| Column_1 | int(255 ).  | NO   |     | NULL    |       |
| Column_2 | varchar(50) | NO   |     | NULL    |       |
| Column_3 | varchar(50) | NO   |     | NULL    |       |
| Column_4 | varchar(50) | NO   |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.02 sec)

mysql>
```

**Appendix F: Explain select command**

```
mysql> explain select Column_1 from test.Table_1 where Column_1 > 0 ;
+----+-------------+---------+------+---------------+------+---------+------
+------+-------------+
| id | select_type | table   | type | possible_keys | key  | key_len | ref  |
rows | Extra       |
+----+-------------+---------+------+---------------+------+---------+------
+------+-------------+
|  1 | SIMPLE      | Table_1 | ALL  | NULL          | NULL | NULL    | NULL |
      2  | Using where |
+----+-------------+---------+------+---------------+------+---------+------
+------+-------------+
1 row in set (0.00 sec)

mysql>
```

**Appendix G: MySQL Pretest Configuration**

| **Table** | **Primary Key** | **Engine** | **Distinct Rows** | **No of rows** |
|---|---|---|---|---|
| Table_1 | No | InnoDB | Column_1 100%<br>Column_2 100%<br>Column_3 79%<br>Column_4 90% | 5 million |
| Table_2 | Yes | InnoDB | All 100% | 5 million |
| Table_3 | Yes | InnoDB | All 100% | 5 million |
| Table_4 | No | InnoDB | All 100% | 1 thousand |
| Table_5 | Yes | InnnoDB | All 100% | 1 thousand |
| Table_6 | Yes | InnoDB | All 100% | 1 thousand |
| Table_7 | No | MyISAM | All 100% | 5 million |
| Table_8 | No | InnoDB | Null values in Column_1 | 5 million |
| Table_9 | No | InnoDB | Column_1 100%<br>Column_2 100%<br>Column_3 79%<br>Column_4 90% | 5 million |

## Appendix H: Test Case Descriptions

| | |
|---|---|
| **Test Number** | 1 |
| **Description** | A simple where condition on a unique column on a table with an InnoDB engine no index and 5 million rows |
| **SQL Syntax** | select column_1 from table_1 where column_1 = 5000; |

| | |
|---|---|
| **Test Number** | 2 |
| **Description** | A simple where condition on a non unique column on (80% values) a table with an InnoDB engine with no index and 5 million rows |
| **SQL Syntax** | select column_3 from table_1 where column_3 = “test_1”; |

| | |
|---|---|
| **Test Number** | 3 |
| **Description** | A simple where condition on a non unique column (90% values unique) on a table with an InnoDB engine with no index and 5 million rows |
| **SQL Syntax** | select column_4 from table_1 where column_4 = “test_1”; |

| | |
|---|---|
| **Test Number** | 4 |
| **Description** | A range query on a Table with an InnoDB engine non unique column (90% values unique) on a table with no index and 5 million rows |
| **SQL Syntax** | select column_1 from table_1 where column_1 between 30000 and 40000; |

| | |
|---|---|
| **Test Number** | 5 |
| **Description** | A range query on a Table with an MyISAM engine non unique column (90% values unique) on a table with no index and 5 million rows |
| **SQL Syntax** | select column_3 from table_7 where column_1 = 5000; |

| | |
|---|---|
| **Test Number** | 6 |
| **Description** | A simple where condition query on a Table with an MyISAM engine non unique column (90% values unique) on a table with no index and 5 million rows |
| **SQL Syntax** | select column_1 from table_7 where column_1 between 30000 and 40000; |

| | |
|---|---|
| **Test Number** | 7 |
| **Description** | A left join condition between two column on a table with no index and 5 million rows |
| **SQL Syntax** | `SELECT table_3.column_1`<br>`FROM table_3`<br>`LEFT JOIN table_2`<br>`ON table_3.column_1 = table_2.column_2;` |

| **Test Number** | 8 |
|---|---|
| **Description** | A simple select on a table with 1000 rows and no index |
| **SQL Syntax** | select column_1 from table_4 where column_1 = 900; |

| **Test Number** | 9 |
|---|---|
| **Description** | A simple select on a table with 1000 rows and unique index |
| **SQL Syntax** | select column_1 from table_5 where column_1 = 900; |

| **Test Number** | 10 |
|---|---|
| **Description** | An update operation on a unique column on a table with an InnoDB engine no index and 5 million rows |
| **SQL Syntax** | update table_1 set column_1 = 34000 where column_1 = 34000 |

| **Test Number** | 11 |
|---|---|
| **Description** | An update operation on a unique column on a table with an InnoDB engine an index and 5 million rows |
| **SQL Syntax** | update table_2 set column_1 = 34000 where column_1 = 34000 |

| **Test Number** | 12 |
|---|---|
| **Description** | An insert operation into table_1; |
| **SQL Syntax** | insert into table_1 values ( '${__Random(10000000,60000006,var)}', 'insert test', 'insert', 'insert' ); |

| **Test Number** | 13 |
|---|---|
| **Description** | An insert operation into table_2; |
| **SQL Syntax** | insert into table_2 values ( '${__Random(10000000,60000006,var)}', 'insert test', 'insert', 'insert' ); |

| **Test Number** | 14 |
|---|---|
| **Description** | An insert operation into table_4; |
| **SQL Syntax** | insert into table_4 values ( '${__Random(10000000,60000006,var)}', 'insert test', 'insert', 'insert' ); |

| | |
|---|---|
| **Test Number** | 15 |
| **Description** | An insert operation into table_5; |
| **SQL Syntax** | insert into table_5 values ( '${__Random(10000000,60000006,var)}', 'insert test', 'insert', 'insert' ); |

| | |
|---|---|
| **Test Number** | 16 |
| **Description** | An insert operation into table_6; |
| **SQL Syntax** | insert into table_6 values ( '${__Random(10000000,60000006,var)}', 'insert test', 'insert', 'insert' ); |

| | |
|---|---|
| **Test Number** | 17 |
| **Description** | An insert operation into table_7; |
| **SQL Syntax** | insert into table_7 values ( '${__Random(10000000,60000006,var)}', 'insert test', 'insert', 'insert' ); |

| | |
|---|---|
| **Test Number** | 18 |
| **Description** | A simple where condition on a unique column (with null values) on a table with an InnoDB engine no index and 5 million rows |
| **SQL Syntax** | select column_1 from table_8 where column_1 = 5000; |

| | |
|---|---|
| **Test Number** | 19 |
| **Description** | A simple where condition on a unique column (with null values) on a table with an InnoDB engine no index and 5 million rows |
| **SQL Syntax** | select column_3 from table_9 where column_1 = 320000 and column_2 like ‘kev%’; |

## Appendix I: Hardware Configuration

| | Value |
|---|---|
| CPU | Intel(R) Core(TM)2 Duo CPU |
| Processor Speed | 2.53 Ghz |
| Memory | 4096 Mb Ram |
| Operating System | Windows 7 |
| MySQL Version | 5.1.43-community |

**Appendix J: Pretest Results**

| Label | # Samples | Average | Median | 90% Line | Min | Max | Error % | Throughput | KB/sec |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 200 | 45735 | 41340 | 74428 | 8990 | 169843 | 0 | 0.016762 | 0.000212799 |
| 2 | 200 | 48493 | 43360 | 85272 | 9141 | 171602 | 0 | 0.016628 | 0.000129908 |
| 3 | 200 | 49333 | 47010 | 83630 | 9241 | 136850 | 0 | 0.016716 | 0.000326494 |
| 4 | 200 | 48143 | 44892 | 78545 | 9073 | 126298 | 0 | 0.016817 | 0.985617246 |
| 5 | 200 | 37484 | 34217 | 63324 | 1080 | 114381 | 0 | 0.016941 | 0.000264697 |
| 6 | 200 | 33905 | 32167 | 57907 | 1121 | 125824 | 0 | 0.017016 | 0.997270298 |
| 7 | 200 | 32504 | 30516 | 57883 | 58 | 106877 | 0 | 0.017003 | 1.807991028 |
| 8 | 200 | 30366 | 29526 | 53128 | 1 | 96733 | 0 | 0.016937 | 0.000198483 |
| 9 | 200 | 26582 | 23212 | 52640 | 0 | 135106 | 0 | 0.016693 | 0.000195624 |
| 10 | 200 | 35488 | 33340 | 67038 | 10615 | 118705 | 0 | 0.016638 | 0.000146235 |
| 11 | 200 | 33196 | 30433 | 57387 | 32 | 142300 | 0 | 0.01664 | 0.000146246 |
| 12 | 200 | 28965 | 30206 | 52021 | 1 | 143710 | 0 | 0.01666 | 0.000146426 |
| 13 | 200 | 28083 | 27859 | 51153 | 1 | 125480 | 0 | 0.016658 | 0.000146407 |
| 14 | 200 | 27983 | 28711 | 55348 | 1 | 121957 | 0 | 0.016589 | 0.000145798 |
| 15 | 200 | 25323 | 21511 | 56404 | 1 | 135714 | 0 | 0.016589 | 0.000145798 |
| 16 | 200 | 22101 | 21938 | 50812 | 1 | 84881 | 0.02 | 0.016589 | 0.000142883 |
| 17 | 200 | 27723 | 27741 | 57932 | 0 | 107845 | 0 | 0.016591 | 0.000145819 |
| 18 | 200 | 34779 | 37184 | 63353 | 8346 | 114253 | 0 | 0.01651 | 0.000209597 |
| TOTAL | 3600 | 34338 | 32083 | 64378 | 0 | 171602 | 0.00109 | 0.291177 | 3.679649927 |

## Appendix K: JMySQLTune Recommendations

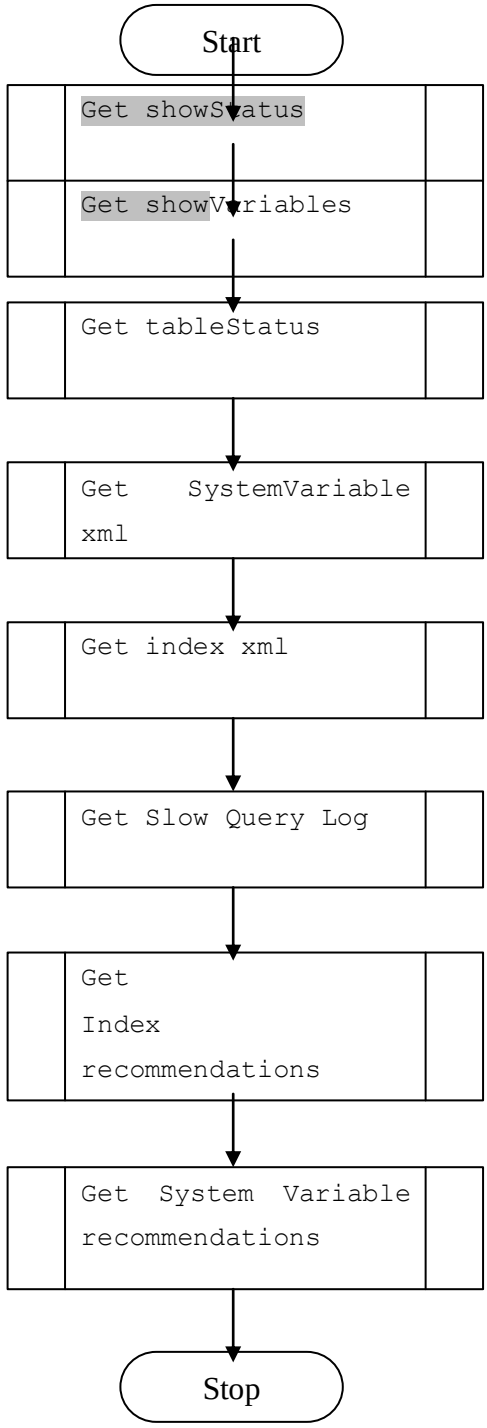
```
             MySQL Database Version
-------------------------------------------------------------
Version                 :- 5.1.43-community-log



             JMySQLTune Indexing Report
-------------------------------------------------------------------
Table                   :- table_8
Column                  :- column_1
Recommendation          :- BTREE UNIQUE INDEX

Table                   :- table_3
Column                  :- column_1
Recommendation          :- PRIMARY KEY COLUMN BTREE UNIQUE INDEX

Table                   :- table_2
Column                  :- column_1
Recommendation          :- PRIMARY KEY COLUMN BTREE UNIQUE INDEX

Table                   :- table_2
Column                  :- column_1
Recommendation          :- PRIMARY KEY COLUMN BTREE UNIQUE INDEX

Table                   :- table_1
Column                  :- column_1
Recommendation          :- PRIMARY KEY COLUMN BTREE UNIQUE INDEX

Table                   :- table_7
Column                  :- column_1
Recommendation          :- RTREE PRIMARY INDEX

Table                   :- table_1
Column                  :- column_3
Recommendation          :- BTREE NON UNIQUE INDEX



             JMySQLTune System Variable Report
-----------------------------------------------------------
System Variable         :- thread cache
Recommendation          :- ok

System Variable         :- aborted connection
Recommendation          :- aborted connections ok

System Variable         :- table locks
Recommendation          :- Table_locks_immediate ok

System Variable         :- query cache
```

```
Recommendation          :- less than 10% of selects

System Variable         :- key cache
Recommendation          :- key cache ok

System Variable         :- key buffers
Recommendation          :- ok

System Variable         :- slow queries
Recommendation          :- ok

System Variable         :- max used connections
Recommendation          :- ok
```

**Appendix L: Post test Results**

| Label | # Samples | Average | Median | 90% Line | Min | Max | Error % | Throughput | KB/sec |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 200 | 76 | 21 | 188 | 0 | 673 | 0 | 7.00035 | 0.088872 |
| 2 | 200 | 66 | 18 | 185 | 0 | 768 | 0 | 7.025679 | 0.054888 |
| 3 | 200 | 61 | 24 | 177 | 0 | 476 | 0 | 7.033585 | 0.137375 |
| 4 | 200 | 83 | 27 | 193 | 15 | 854 | 0 | 7.037298 | 412.4379 |
| 5 | 200 | 73 | 24 | 204 | 0 | 975 | 0 | 7.068137 | 0.11044 |
| 6 | 200 | 76 | 42 | 185 | 13 | 534 | 0 | 7.065141 | 414.0697 |
| 7 | 200 | 150 | 121 | 275 | 56 | 646 | 0 | 7.05791 | 750.4819 |
| 8 | 200 | 91 | 63 | 230 | 0 | 1015 | 0 | 7.216048 | 0.084563 |
| 9 | 200 | 59 | 17 | 172 | 0 | 739 | 0 | 7.233535 | 0.084768 |
| 10 | 200 | 64 | 23 | 186 | 0 | 535 | 0 | 7.235628 | 0.063594 |
| 11 | 200 | 59 | 5 | 173 | 0 | 1110 | 0 | 7.269555 | 0.063893 |
| 12 | 200 | 68 | 19 | 171 | 1 | 1011 | 0 | 7.269291 | 0.06389 |
| 13 | 200 | 66 | 13 | 180 | 0 | 770 | 0 | 7.28863 | 0.06406 |
| 14 | 200 | 72 | 18 | 187 | 1 | 1015 | 0 | 7.370555 | 0.06478 |
| 15 | 200 | 62 | 10 | 168 | 0 | 955 | 0 | 7.417022 | 0.065189 |
| 16 | 200 | 63 | 11 | 196 | 0 | 1021 | 0 | 7.452953 | 0.061574 |
| 17 | 200 | 54 | 9 | 153 | 0 | 522 | 0 | 7.498219 | 0.065902 |
| 18 | 200 | 58 | 14 | 148 | 0 | 747 | 0 | 7.507789 | 0.095314 |
| TOTAL | 3600 | 72 | 24 | 188 | 0 | 1110 | 0 | 125.4443 | 1559.053 |

**Appendix M: JMySQLTune Main Flow Chart**



## Appendix N: JMySQLTune Main Source Code

```
/**
 * JMySQLTuner is the base class for a product called JMySQLTune
 *
 * This is an automated database tuning tool which will be used
 * to solve the problems of identifying effective recommendations
 * for index and system variable selection in MySQL community servers
 * using scientific processes and repeatable techniques.
 * which allow an application to draw onto components realized on
 * various devices or onto off-screen images.
 *
 * @author      Kevin Spillane
 * @version     1, 0
 * @since       1.0
 *
 */
import java.io.File;
import java.util.List;
import javax.xml.stream.XMLStreamException;
import javax.swing.JFrame;


public class JMySQLTuner {

public static void main(String args[]) throws XMLStreamException
  {

                ShowVariables showEntry;



                /*
                 * Database Section
                 *
                 * This section of code is concerned with
                 * logging into the MySQL database and extracting some configuration information
                 *
                 *    Set Up Query Connection to database
                 *    Read System Variable Tuning XML File
                 *    Read Index Tuning XML File
                 *    Execute SQL Show Variables command
                 *    Execute SQL Show Status Command
                 *
                 *    Parse the System Variable XML file
                 *
                 */
                JMySQLTuneQueries connect = new JMySQLTuneQueries( "root", "test", "jdbc:mysql://localhost/test", "test");
                JMySQLTuneXML decisionxml = new JMySQLTuneXML("C:\\Users\\spillke\\Documents\\Masters\\thesis\\JMySQLTuneSystemVariable.xml");
                JMySQLTuneXML indexml = new JMySQLTuneXML("C:\\Users\\spillke\\Documents\\Masters\\thesis\\JMySQLTuneIndexNew.xml");


                File logfile = new File( "C:\\shspillke1-slow.log" );

                List<ShowVariables> showStatus = connect.getshowStatus();
                List<ShowVariables> showVariables = connect.getshowVariables();
                List<TableStatus> tableStatus = connect.getshowtableStatus();
                List<JMySQLXML> decisiontree = decisionxml.getXMLParser();
                List<JMySQLXML> decisionindex = indexml.getXMLParser();
                List<JMySQLSlowLog> slowQueries = connect.findSlowQueries(logfile);

    /*
     * Display Version Information
     */
                for ( int h = 0; h < showVariables.size(); h++ )
                {
                                showEntry = showVariables.get(h);

                                if ( showEntry.getVariableName().matches("version"))
                                {
                                                System.out.printf("            MySQL Database Version\n-------------------------------------------------------------\n");
                                                System.out.printf(
                                                                "Version                :- %s\n\n\n", showEntry.getVariableValue());
                                }
                }

                System.out.printf("\n\n            JMySQLTune Indexing Report\n-------------------------------------------------------------\n");
                JMySQLGetDecision queries = new JMySQLGetDecision( slowQueries, decisionindex, connect, tableStatus, showVariables, showStatus );
                queries.getIndex(slowQueries, decisionindex, connect, tableStatus);
```

```
            System.out.printf("\n\n            JMySQLTune System Variable Report\n----------------------------------------------------------------\n");
            queries.getSystemVariables(slowQueries, decisiontree, connect, tableStatus, showVariables, showStatus );

            /*
             * close the database connection
             */
            connect.close();
    }

}
```

## ANNOTATED BIBLIOGRAPHY

Agrawal, R., Ailamaki, A., Bernstein, P. A., Brewer, E. A., Carey, M. J., Chaudhuri, S., el al.,(2008). The Claremont report on database research. *SIGMOD Rec.*, *37*(3), 9-19. doi: 10.1145/1462571.1462573.

The authors are a group of researchers, architects, users and pundits met at the Claremont Resort in Berkeley, California to discuss the state of database research and its impacts. This report details this meeting, during which a wide variety of issues were discussed. The author's evaluated future research areas, which they deemed to include new database engine architectures, declarative programming languages, the interplay of structured and unstructured data, cloud data services and mobile and virtual worlds. Finally they discuss other database-related topics, such as the changes in community processes, to move the research agenda forward.

Agrawal, S., Chaudhuri, S., Kollar, L., Marathe, A., Narasayya, V., & Syamala, M (2005). Database tuning advisor for microsoft SQL server 2005: demo. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 930-932). Baltimore, Maryland: ACM. doi: 10.1145/1066157.1066292.

The authors of this paper discuss Database Tuning Advisor (DTA) a physical database design tool, which is part of Microsoft's SQL Server 2005 relational database management system. The authors illustrate how the DTA is used to meet the performance and manageability requirements

of DBA's. They provide experimental research to illustrate, how efficient the DTA is in producing reliable recommendations for indexes, materialized views and horizontal partitioning.

Ahmed, R., Lee, A., Witkowski, A., Das, D., Su, H., Zait, M., el al.,(2006). Cost-based query transformation in Oracle. In *Proceedings of the 32nd international conference on Very large data bases* (pp. 1026-1036). Seoul, Korea: VLDB Endowment. Retrieved July 29, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1182635.1164215&coll=Portal&dl=ACM&CFID=45662376&CFTOKEN=73452892

A description of cost-based query transformation approach for the physical tuning of Oracle databases, is undertaken by the authors in this paper. They assess the requirement for this framework. They also evaluate possible interactions among some of the transformation and efficient algorithms for enumerating the search space of cost-based transformations. The authors then describe a practical technique to combine cost-based transformations with Oracles physical optimizer. Finally they offer evidence that their approach produces significant execution time benefits.

Annesley, C (2006). Open source database 'years away from being a serious contender'. *Computer Weekly*,

In this article the authors compare open source databases with their commercial rivals. They provide evidence suggesting that open source databases such as MySQL, Cloudscape and Firebird are still years away from offering a viable, enterprise-ready alternative to their

commercial competitors. According to the authors, although open source operating systems are used widely by organisations to run databases, the development of open source databases is not so advanced.

Bayan, M., & Cangussu, J. W (2008). Automatic feedback, control-based, stress and load testing. In *Proceedings of the 2008 ACM symposium on Applied computing* (pp. 661-666). Fortaleza, Ceara, Brazil: ACM. doi: 10.1145/1363686.1363847.

Authors of this article suggest a new method for stress and load testing. This method is based on the use of a PID controller to automatically drive inputs. This is done to achieve a pre-specified level of stress/load for resources of interest. The authors, illustrate, how current approaches present limitations with respect to automation and applicability. Finally they perform experimental research, presented as evidence of the applicability and accuracy of their approach.

Benoit, D. G (2005). Automatic Diagnosis of Performance Problems in Database Management Systems. In *Proceedings of the Second International Conference on Automatic Computing* (pp. 326-327). IEEE Computer Society. Retrieved August 13, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1078486&dl=ACM&coll=Portal&CFID=48487895&CFTOKEN=47747821

The above author discusses the relationship between database and DBMS resource allocation. These relationships often make problem diagnosis and performance tuning difficult, time-

consuming tasks. The author provides details of a diagnosis model which may be used to automatically identify performance bottlenecks. Through experimental research the author provides evidence that the proposed technique has the potential to both, increase performance and reduce total cost of ownership of the database.

Bruno, N., & Chaudhuri, S (2005). Automatic physical database tuning: a relaxation-based approach. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 227-238). Baltimore, Maryland: ACM. doi: 10.1145/1066157.1066184.

In this paper the authors present the relaxation-based approach to physical database tuning. They commence by examining the architecture of competing approaches, which they suggest mostly consists of, candidate access paths heuristically chosen based on the structure of each input query with a search performed to identify an optimal configuration. From this research the authors the authors note, that most recent techniques have become increasingly complex which makes it very difficult to analyze and extract properties. Finally the authors provide a new framework for physical design problem which they claim significantly reduces the assumptions and heuristics used in competitors solutions.

Bruno, N., & Chaudhuri, S (2006). To tune or not to tune?: a lightweight physical design alerter. In *Proceedings of the 32nd international conference on Very large data bases* (pp. 499-510). Seoul, Korea: VLDB Endowment. Retrieved July 24, 2009, from

http://portal.acm.org.dml.regis.edu/citation.cfm?id=1182635.1164171&coll=Portal&dl=ACM&CFID=46443895&CFTOKEN=57659103

This paper provides details of an alerter that can be used by the DBA to determine when a physical design tool should be invoked. The authors claim this alerter is of benefit, as it is a lightweight mechanism that guarantees database performance improvement by alerting the DBA as to the correct moment to invoke the alerter. Finally, the authors illustrate, through experiment, how the alerter handles large workloads with little overhead.

Bruno, N., & Chaudhuri, S (2007a). Online autoadmin: (physical design tuning). In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (pp. 1067-1069). Beijing, China: ACM. doi: 10.1145/1247480.1247608.

The author's in this paper criticize automated physical design solutions. They believe they require explicit invocations of tuning tools and critically depend on DBA's gathering representative workloads manually. To resolve this issue, they discuss a monitoring/tuning DBMS component. They suggest that this component be "always on" continuously modifies the current physical design reacting to varying workload or data characteristics. The author's prototype this solution in Microsoft SQL Server 2005 and conclude the solution imposes low overhead and takes into account storage constraints, update statements and the cost to create physical structures.

Bruno, N., & Chaudhuri, S (2007b). Physical design refinement: The \‘merge-reduce\’ approach. *ACM Trans. Database Syst., 32*(4), 28. doi: 10.1145/1292609.1292618.

In this paper, the authors provide criticise current physical database design tools. The authors conclude these tools do not provide adequate support for the incremental and flexible refinement of existing physical structures. Further in this article, the authors propose a transformational framework, based on primitive operations, merging and reduction. The author's hypothesis is there operations, can refine a configuration, by treating indexes and materialized views in a unified way.

Bujdei, C., Moraru, S., & Dan, S (2008). Optimize databases for health monitoring systems. In *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments* (pp. 1-8). Athens, Greece: ACM. doi: 10.1145/1389586.1389676.

The authors study and describe methods which they hypothesize could be used for optimizing a database. The authors consider a number optimizing possibilities for a user wireless sensor networks (WSN) inside the systems dedicated for monitoring the health status of the patients. The authors conclude, these kinds of optimizations could also be implemented on other type of similar systems.

Cabral, S. K., & Keith Murphy ( (2009). *MySQL Administrator's Bible* (p. 888). Wiley Publishing. Retrieved January 7, 2010, from http://portal.acm.org/citation.cfm?id=1643594.

In this book, the authors provide a reference manual to MySQL database systems. This book provides discusses the fundamentals of MySQL database management. Topics discussed include SQL queries, data and index types, stored procedures, functions and triggers. The authors also present coverage of maintenance topics such as server tuning, managing storage engines, caching, backup and recovery, performance monitoring and security.

Chaudhuri, S., & Narasayya, V (1998). AutoAdmin \“what-if\” index analysis utility. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (pp. 367-378). Seattle, Washington, United States: ACM. doi: 10.1145/276304.276337.

This paper discusses the importance of automatic index selection in reducing the overhead of database administration. The authors discuss the complexity for database administrator (DBA) to be able to perform a quantitative analysis of the existing indexes, whilst also having to cognisant of the hypothetical indexes. The authors explain the complexity suggesting analysis consists of analyzing workloads over the database, estimating changes in the cost of a workload and studying index usage while taking into account projected changes in the sizes of the database tables Finally the authors describe, a novel index analysis utility which they have prototyped for Microsoft SQL Server 7.0

Chaudhuri, S., & Narasayya, V (2007). Self-tuning database systems: a decade of progress. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 3-14). Vienna, Austria: VLDB Endowment. Retrieved July 29, 2009, from

http://portal.acm.org.dml.regis.edu/citation.cfm?id=1325851.1325856&coll=Portal&dl=ACM&CFID=45688813&CFTOKEN=83576216

The main topic of interest in this paper is the advantages of self-tuning database systems over the years 1997 to 2007. The authors focus on the problem associated with automated physical database design. They also review areas where research on self-tuning database technology has made significant progress.

Chaudhuri, S., & Weikum, G (2005). Foundations of automated database tuning. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (pp. 964-965). Baltimore, Maryland: ACM. doi: 10.1145/1066157.1066305.

Automated database tuning is explored in this paper by Chaudhuri and Weikum. They note most DBA's are presented with a number challenges. These include, reducing total cost of ownership and managing information systems infrastructure in a cost-effective manner is a growing challenge. The authors note total cost of ownership (TCO) of information technology is increasingly dominated by people costs. They also believe human error, is the single most reasons for system outage and unacceptable performance in many database systems. The authors conclude that these factors combine to make automated database tuning an essential task.

Chung, I., & Hollingsworth, J. K (2004). Using Information from Prior Runs to Improve Automated Tuning Systems. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing* (p. 30).

IEEE Computer Society. Retrieved August 20, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1048933.1049974&coll=Portal&dl=ACM&CFID=48157413&CFTOKEN=58734191

In this paper the authors present a parameter prioritizing tool. The authors hypothesis is based on the importance of historical data in speeding up the tuning process. They propose that this data be used to identify the critical parameters for performance. Furthermore they verify their hypothesis using synthetic data and later on using a real cluster-based web service system. They conclude the framework has two benefits; firstly it reduces the time spent tuning from 35% up to 50% and secondly it reduces the large variations in performance while tuning.

Comer, D (1978). The difficulty of optimum index selection. *ACM Trans. Database Syst., 3*(4), 440-445. doi: 10.1145/320289.320296.

The difficulty associated with index selection is investigated in this paper. The authors refer to past research which to index has considered various storage structures and access assumptions. They investigate if a simpler algorithm might exist. However they conclude even under a simple cost criterion the problem is computationally difficult in a precise sense.

Dageville, B., Das, D., Dias, K., Yagoub, K., Zait, M., & Ziauddin, M (2004). Automatic SQL tuning in oracle 10g. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* (pp. 1098-1109). Toronto, Canada: VLDB Endowment. Retrieved July 31, 2009,

from
http://portal.acm.org.dml.regis.edu/citation.cfm?id=1316689.1316784&coll=Portal&dl=GUIDE&CFID=47099211&CFTOKEN=59625626

In this paper the authors provide, an explanation, for the importance of automatic SQL tuning in Oracle 10g. Without this feature, the authors speculate DBA/programmer must have expertise in several domains, including query optimization, access design and SQL design. The authors also note that manual SQL tuning is a time consuming task due to the large volume and evolving nature of the SQL workload and its underlying data.

Dageville, B., & Zait, M (2002). SQL memory management in Oracle9i. In *Proceedings of the 28th international conference on Very Large Data Bases* (pp. 962-973). Hong Kong, China: VLDB Endowment. Retrieved August 10, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1287454&dl=ACM&coll=Portal&CFID=48190571&CFTOKEN=72295952

In this paper, the authors discuss how Oracle 9i operates with complex database queries. These operations often require the use of memory-intensive operators like sort and hash-join. Those operators need memory, also referred to as SQL memory, to process their input data. The author's discusses Oracle 9i strategy to manage this memory.

Di Giacomo (2005, May 1). MySQL: Lessons Learned on a Digital Library.. Retrieved July 28, 2009, from http://www2.computer.org/portal/web/csdl/doi/10.1109/MS.2005.71

In this paper the author review's advances in open source databases. The author hypothesis is these products are achieving enterprise-level quality. They provide some evidence citing the Los Alamos National Laboratory's Research Library selecting the open source MySQL database management system to develop a comprehensive database of scientific journal articles and citation information. The authors conclude that lessons learned should help advocates promote MySQL and open source for building a sound technology infrastructure.

Elnaffar, S. S (2002). A methodology for auto-recognizing DBMS workloads. In *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research* (p. 2). Toronto, Ontario, Canada: IBM Press. Retrieved August 20, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=782115.782117&coll=Portal&dl=ACM&CFID=48115736&CFTOKEN=15655660

In this paper Elnaffar considers the difference in tuning database systems based on workload. They state allocations of resources such as main memory are different workload type is Online Transaction Processing (OLTP) or Decision Support System (DSS). Therefore their hypothesis is database administrators must recognize the workload type in order to maintain acceptable levels of performance. The author's provide a classification model prototype, based on the most significant workload characteristics that differentiate OLTP from DSS. Finally they some experiments, which illustrate the workload classifiers are reliable and have high accuracy in recognizing the type of the workload

.

Finkelstein, S., Schkolnick, M., & Tiberio, P (1988). Physical database design for relational databases. *ACM Trans. Database Syst., 13*(1), 91-128. doi: 10.1145/42201.42205.

In this paper, the author's describe the concepts used in the implementation of DBDSGN. This is a physical design tool for relational databases developed at the IBM San Jose Research Laboratory. This tool suggests physical configurations for efficient performance should be based on workload. This tool, only evaluates workload statements for atomic configurations of indices, which have only one index per table, to finds efficient solutions to the index-selection problem. The authors also discuss the possibility of using heuristics, to reduce execution time.

Hammer, M., & Chan, A (1976). Index selection in a self-adaptive data base management system. In *Proceedings of the 1976 ACM SIGMOD international conference on Management of data* (pp. 1-8). Washington, D.C.: ACM. Retrieved August 25, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=509383.509385&coll=Portal&dl=ACM&CFID=48811291&CFTOKEN=68747452

In this paper, the author's the address the problem of automatically adjusting the physical organization of a database to optimize its performance. This is early research into automatic index selection. They use exponential smoothing techniques for averaging statistics in order to predict future characteristics. A heuristic algorithm for selecting indices based on these results is then presented.

He, Z., Lee, B. S., & Snapp, R (2005). Self-tuning cost modeling of user-defined functions in an object-relational DBMS. *ACM Trans. Database Syst., 30*(3), 812-853. doi: 10.1145/1093382.1093387.

In this paper the author's hypothesis is there is a need to develop an execution cost model for query optimizers. The authors believe existing static approaches cannot adapt to changing User Defined Functions UDF execution patterns and thus degrade in accuracy. In response to this the authors propose a new approach based on the self-tuning DBMS by which the cost model is maintained dynamically and incrementally as UDFs are being executed online.

Ip, M., Saxton, L., & Raghavan, V (1983). On the Selection of an Optimal Set of Indexes. *Software Engineering, IEEE Transactions on, SE-9*(2), 135-143.

The authors of this article present a probabilistic model of transactions (queries, updates, insertions and deletions) to a file. They develop an evaluation function, based on the cost saving attributable, to determine an index set. The authors believe the maximization of this function should provide an optimal index set. Finally the authors attempt to prove their theory through experiment.

Ko, B., Lee, K., Amiri, K., & Calo, S (2003). Scalable Service Differentiation in a Shared Storage Cache. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*

(p. 184). IEEE Computer Society. Retrieved January 5, 2010, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=850929.851963&coll=Portal&dl=GUIDE&CFID=69772859&CFTOKEN=67193021

In this paper the authors discuss Quality of Service (QoS) architecture for shared storage proxy cache. They hypothesis is this approach, provides long-term hit rate assurances to competing classes. They propose an architecture which consists of per-class feedback controllers which is used to track the performance of each class, a fairness controller that reallocates excess resources and a contention resolver used to decide cache allocation in the case when at least one class does not meet its target hit rate. From this research the authors provide guidelines for designing QoS mechanisms.

Kormilitsin, M., Chirkova, R., Fathi, Y., & Stallmann, M (2008). View and index selection for query-performance improvement: quality-centered algorithms and heuristics. In *Proceeding of the 17th ACM conference on Information and knowledge management* (pp. 1329-1330). Napa Valley, California, USA: ACM. doi: 10.1145/1458082.1458261.

In this paper the authors develop a methodology that delivers user-specified global optimum performance of views and indexes for any given workload. The authors use experimental results and comparisons on synthetic and benchmark instances to illustrate the suitability of this approach. From this research the authors provide evidence that their methodology is successful in its aims of improving performance.

Manegold, S., Boncz, P. A., & Kersten, M. L (2000). Optimizing database architecture for the new bottleneck: memory access. *The VLDB Journal*, *9*(3), 231-246.

In this paper, the authors discuss how main-memory access is increasingly being perceived as a performance bottleneck for database systems. In this article, they propose a simple scan test to demonstrate the impact of this bottleneck. They use the results of these tests, to provide guidelines for database architecture, for both data structures and algorithms. They discuss the use of vertically fragmented data structures optimize cache performance on sequential data access. They also analyze equi-joins, random-access operations and radix algorithms for partitioned hash-join. Using experimental research they validate a detailed analytical model that incorporates memory access cost.

Moussa, M., Ruwanpura, J. Y., & Jergeas, G (2004). Decision tree module within decision support simulation system. In *Proceedings of the 36th conference on Winter simulation* (pp. 1268-1276). Washington, D.C.: Winter Simulation Conference. Retrieved September 15, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1161734.1161967&coll=Portal&dl=GUIDE&CFID=52939072&CFTOKEN=73692752.

In this article the authors discuss the benefits of decision trees in problem analysis. They propose a new application of a Special Purpose Simulation (SPS) program in developing a

Decision Tree module that is part of a unified Decision Support System (DSS) template. This template consists of three modules: Decision Tree (DT), shortest and longest path Dynamic Programming (DP) Network and Cost / Time (CT) Estimate network. They integrate the Decision Tree model with other modules and allow users to model decision trees with variables that are based on probabilistic random numbers.

Osogami, T., & Kato, S (2007). Optimizing system configurations quickly by guessing at the performance. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (pp. 145-156). San Diego, California, USA: ACM. doi: 10.1145/1254882.1254899.

In this paper the authors discuss the problem associated with finding the optimal tuning configuration, finding that it is time-consuming task. This leads the authors to propose an algorithm, called Quick Optimization via Guessing (QOG), which selects nearly best configurations with high probability. The author's hypothesis is based around two principles, the measurement of a configuration is terminated as soon as the configuration is found to be suboptimal and performance of a configuration is guessed at based on the measured similar configurations. They provide experimental research applying QOG of a real Web system and find a drastic reduction in the total measurement time needed to select the best configuration.

Qiao, L., Soetarman, B., Fuh, G., Pannu, A., Cui, B., Beavin, T., el al.,(2007). A framework for enforcing application policies in database systems. In *Proceedings of the 2007 ACM SIGMOD*

*international conference on Management of data* (pp. 981-992). Beijing, China: ACM. doi: 10.1145/1247480.1247597.

In this paper discuss the issues surrounding the administrator of modern databases. The author discussed how these factors combined with a scarcity of skilled database professionals has meant that human costs have begun to dominate the total cost of ownership (TCO) of a database system. In response to this, the authors discuss the pressure on commercial database vendors to provide solutions which make their products easy to administer in areas such as problem diagnostics, monitoring, query tuning, access control and system configuration.

Schnaitter, K., Abiteboul, S., Milo, T., & Polyzotis, N (2006). COLT: continuous on-line tuning. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (pp. 793-795). Chicago, IL, USA: ACM. doi: 10.1145/1142473.1142592

In this paper the authors introduce a novel database self tuning framework named COLT (Continuous On-Line Tuning). The authors propose this framework continuously monitor the incoming queries and adjusts the system configuration in order to maximize query performance. The hypothesis is the framework, gathers performance statistics at different levels of detail and then allocates resources to the most promising candidate configurations. The framework will use heuristics to regulate its own performance. The authors use experiment to demonstrate the framework in a PostgreSQL database system, showing the internal operation of COLT and the adaptive selection of indices.

Slivinskas, G., Jensen, C. S., & Snodgrass, R. T (2001). Adaptable query optimization and evaluation in temporal middleware. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data* (pp. 127-138). Santa Barbara, California, United States: ACM. doi: 10.1145/375663.375678.

This article describes the architecture and implementation of the temporal middleware component, termed TANGO. This component is based on the Volcano extensible query optimizer and the XXL query processing library. The hypothesis is that TANGO accepts temporal SQL statements and produces a corresponding query plan consisting of algebraic as well as regular SQL parts. The middleware component also uses performance feedback from the DBMS to adapt its partitioning of subsequent queries into middleware and DBMS parts. The authors provide evidence through experiment of Tango's internal processing capability and its cost-based mechanism for apportioning the processing between the middleware and the underlying DBMS.

Sockut, G. H., & Iyer, B. R (2009). Online reorganization of databases. *ACM Comput. Surv.*, *41*(3), 1-136. doi: 10.1145/1541880.1541881.

The authors discuss the need for reorganization of databases throughout its life. They define reorganization as the change in some aspect of the logical and/or physical arrangement of a database. According to the authors the traditional approach of a database offline can be unacceptable for a highly available (24-hour) database. This article presents the authors views on requirements, issues and strategies for online reorganization. The authors analyze issues and

present strategies, which use the issues. The issues mostly involve design trade-offs. The article surveys online strategies in three categories of reorganization. These categories include maintenance, changing the physical database definition and changing the logical database definition.

Storm, A. J., Garcia-Arellano, C., Lightstone, S. S., Diao, Y., & Surendra, M (2006). Adaptive self-tuning memory in DB2. In *Proceedings of the 32nd international conference on Very large data bases* (pp. 1081-1092). Seoul, Korea: VLDB Endowment. Retrieved August 27, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1182635.1164220&coll=Portal&dl=ACM&CFID=50224897&CFTOKEN=94379955

A review ofDB2's Self-Tuning Memory Manager (STMM) is undertaken by the authors in this paper. STMM provides adaptive self tuning of database memory heaps and cumulative database memory allocation. The authors state the use of cost-benefit analysis and control theory techniques a makes this technology of particular interest. They note cost-benefit analysis allows STMM to tune memory between different memory resource users such as compiled statement cache, sort and buffer pools. Through research the authors conclude STMM tunes memory allocation to a level approximate to the optimal.

Talebi, Z. A., Chirkova, R., Fathi, Y., & Stallmann, M (2008). Exact and inexact methods for selecting views and indexes for OLAP performance improvement. In *Proceedings of the 11th*

*international conference on Extending database technology: Advances in database technology* (pp. 311-322). Nantes, France: ACM. doi: 10.1145/1353343.1353383.

In this paper, the authors review methods for selecting views and indexes for OLAP databases. In their opinion view and index selection problem can be redefined as an optimization problem, where inputs include data-warehouse schema, data-analysis queries and storage-limit constraints. Secondly the output is a set of views and indexes that minimizes the costs of the input queries, subject to the storage limit.

The authors conclude, while heuristic strategies for choosing views or indexes might help to some extent in improving the costs, there is no known approximation algorithm for OLAP view or index selection with nontrivial performance guarantees.

Tran, D. N., Huynh, P. C., Tay, Y. C., & Tung, A. K. H (2008). A new approach to dynamic self-tuning of database buffers. *Trans. Storage, 4*(1), 1-25. doi: 10.1145/1353452.1353455.

In this paper the authors discuss the need for automated performance tuning of databases. The authors contend increasing complexity coupled with increased complexity makes manual tuning a task performed by performance experts infeasible. The author's hypothesis these factors have combined to lead major vendors to offer tools that automatically and dynamically tune a database system.

Vuduc, R., Demmel, J. W., & Bilmes, J. A (2004). Statistical Models for Empirical Search-Based Performance Tuning. *International Journal of High Performance Computing Applications*, *18*(1), 65-94. doi: 10.1177/1094342004041293.

In this paper the authors commence by discussing the growth in popularity for automatic tuning systems over the last number of years. According to the authors this has emerged in response to the problems associated with manual tuning. The authors believe most tuning approaches generate a large number of possible configurations and then select one based on a combination of heuristic modelling and empirical search. In this paper the author's presents quantitative to determine the value of such an approach to tuning.

Weikum, G., Moenkeberg, A., Hasse, C., & Zabback, P (2002). Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proceedings of the 28th international conference on Very Large Data Bases* (pp. 20-31). Hong Kong, China: VLDB Endowment. Retrieved August 17, 2009, from http://portal.acm.org.dml.regis.edu/citation.cfm?id=1287369.1287373&coll=GUIDE&dl=GUIDE&CFID=48946147&CFTOKEN=60562462

In this paper, the authors reviews and assesses the advances that had been made in self-tuning of database technology between the years 1992 and 2002. The author's notice all major researchers conclude self-tuning database technology should be based on the paradigm of a feedback control loop, mathematical models and their proper engineering into system components. The authors conclude, the composition of information services into truly self-

tuning, higher-level E-services may require a radical departure towards simpler, highly componentized software architectures with narrow interfaces between RISC-style "autonomic" components.

Wiese, D., Rabinovitch, G., Reichert, M., & Arenswald, S (2008). Autonomic tuning expert: a framework for best-practice oriented autonomic database tuning. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds* (pp. 27-41). Ontario, Canada: ACM. doi: 10.1145/1463788.1463792.

In this paper the authors discuss the need for automated performance tuning due to the growth in scale and complexity of databases. High performance, availability and further service level agreements need to be satisfied under any circumstances to please customers. The authors suggest the use of best-practice techniques similar to the ones used in other industries such as medicine may be of benefit to organizations developing such database performance tuning products.

Xu, J (2008). Rule-based automatic software performance diagnosis and improvement. In *Proceedings of the 7th international workshop on Software and performance* (pp. 1-12). Princeton, NJ, USA: ACM. doi: 10.1145/1383559.1383561.

A rule-based framework to identify the root causes of performance limits is investigated in this paper. To achieve this, the authors separate the effects of the system configuration from the limits imposed by the software design. They then propose a framework which applies

transformations to a performance model in order to obtain another improved one. The improvements imply configuration and design changes which may be then applied to the system.

Zaman, M., Surabattula, J., & Gruenwald, L (2004). An auto-indexing technique for databases based on clustering. In *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on* (pp. 776-780). Presented at the Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on.

In this paper, the authors propose an the use of clustering techniques in order to produce automated index selection tool capable of analyzing large amounts of data and suggesting a good set of indexes. The authors provide a description of this novel auto-indexing using clustering technique. Through experimental research they demonstrate that the proposed tool results in better performance than Microsoft SQL server index selection tool.

Zawodny, J. D., & Balling, D. J (2004). *High Performance MySQL* (p. 276). O'Reilly.

In this book the authors discuss a wide range of issues related to building fast, reliable systems with MySQL. The authors dedicate a number of chapters related to MySQL Tuning. The authors state, what they believe to be the most likely source of bottleneck, in a MySQL database system. The authors then provide details on how these bottlenecks may be indentified using tools and utilities contained within MySQL. Finally the authors discuss strategies for removing these bottlenecks, thus improving system performance.

Zenmyo, T (2009). Identifying performance bottlenecks based on the local parameter tuning. In *Proceedings of the 6th international conference on Autonomic computing* (pp. 53-54). Barcelona, Spain: ACM. doi: 10.1145/1555228.1555242.

In this paper, the authors propose a novel method to identify performance bottlenecks together with their root causes. In their technique, each component in the target system is automatically tuned by adjusting configuration parameters. They then evaluate the effect the parameter tuning to diagnose whether the bottleneck exists in the tuned component or not. The author's hypothesis is bottlenecks caused by resource shortages can be identified based on the applicability of the parameter tuning. Therefore parameter tuning is useful to identify the bottlenecks in relational databases.