TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering
Chair of System Software

IAG70LT

Oleg Dmitrijev 121998IASM

SELECTION CRITERIA OF THE DATABASE MANAGEMENT SYSTEM FOR ESTONIAN SMALL AND MEDIUM SIZED ENTERPRISES

Master`s thesis

Supervisor: Vladimir Viies
Associate professor

Tallinn 2015

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: Oleg Dmitrijev

05.12.14

Abstract

The goal of this thesis is to develop a methodology facilitating the process of multi criteria selection of database management system according to needs of Estonian small or medium sized enterprises. The developed methodology is based on mathematical method of decision-making named Analytic Hierarchy Process invented and popularized by mathematician Thomas L. Saaty. It has been proposed to implement open source project named as The Open Source Database Benchmark (GNU GPL), based on ANSI AS^3AP standard, for performance evaluation.

In order to test the viability of the proposed methodology, the evaluation process of selection criteria has been executed with three free database management systems. The criteria and features of selected database management systems have been estimated and compared. In the performance evaluation part, the code of The Open Source Database Benchmark has been adopted regarding to particular features of the database management systems. The projects have been written in C++ programming language and compiled using g++ compiler. All projects' code resides on the DVD attached to the thesis. After the assessment and finding out the most relevant database management system, the further analysis has revealed possible improvements of the methodology.

In recent years due to growing volumes of carbon dioxide emissions and as result global climate change, the issue of power efficiency in all areas of human activity has become actual. Therefore, in this thesis the meaning of power efficiency is stressed and although in common case, other selection criteria can be dominant, the elaboration and analysis of power efficiency evaluation are issued separately.

The thesis is in English and contains 65 pages of text, 3 chapters, 14 figures, 28 tables.

Annotatsioon

Käesoleva magistritöö eesmärk on Eesti väikse ja keskmise ettevõttele sobiva andmebaasi haldamise süsteemi mitmekriteeriumilise hindamise metoodika loomine. Töö raames loodud metoodika baseerub matemaatilisel Analüütiliste Hierarhiate Meetodil. Antud meetod on Ameerika Ühendriikide matemaatika professor Thomas L. Saaty arendatud ja populariseeritud. Analüütiliste Hierarhiate Meetodi kasutus leiab aset finantsturgudel investeeringute hindamisel, kinnisvara arendamisel, maavarade kaevandamisel ja muudel tegevusaladel. Ülalmainitud meetod võimaldab läbi viia võrdlust kasutades kahe kriteeriumi suhtelist hinnangut. Võrdlemisprotsessi käigus saadud hinnanguid paigutatakse võrdlemismaatriksisse, mille peal edasi teostatakse vastavaid matemaatilisi operatsioone. Tulemusena saadakse prioriteetide vektori, mille elementide väärtused ongi otsitavad prioriteetide hinnangud. Andmebaasi haldamise süsteemi jõudluse hindamiseks oli valitud vabavaraline GNU General Public License all levitatud projekt nimega The Open Source Database Benchmark (OSDB). Antud projekt on C++ programmeerimiskeeles kirjutatud ja kehastab ANSI AS^{3}AP standardi autorite ideed andmebaasi haldamise süsteemi jõudluse hindamise kohta. Pakutud hindamise metoodika rakendatavuse testimiseks oli läbi viidud kolme andmebaasi haldamise süsteemi võrdlus ja hinnang. Seoses sellega et OSDB projekti viimane ja kasutusel olnud versioon oli võimeline vaid MySQL, olemasolev kood oli modifitseeritud selleks et hinnata PostgreSQL and FirebirdSQL jõudlust võttes arvesse nende andmebaasi juhtimissüsteemi võimalused ja omapärad. Kõikide projektide programmikood leidub magistritööle lisatud DVD-l. Kasutades Analüütiliste Hierarhiate Meetodi abil välja arvutatud kriteeriumite kaalude prioriteete Eesti väikse ja keskmise ettevõtte jaoks on leitud selle klassi ettevõtetele kõige sobivam andmebaasi juhtimissüsteem. Selle järgnevas analüüsis on pakutud metoodika võimalikud parandused.

Seoses süsihappegaasi osakaalu suurendamisega atmosfääris ja selle negatiivse mõjuga kliimale energiatarve efektiivsus on muutunud aktuaalseks paljudes inimkonna tegevusalades. Selles magistritöös on esile tõstetud andmebaasi haldamise süsteemi energiatarve efektiivsuse hindamise metoodika, kuigi süsteemi üldhindamisel võib leiduda olulisemaid kriteeriume.

Magistritöö on kirjutatud inglise keeles ning sisaldab teksti 65 leheküljel, 3 peatükki, 14 joonist, 28 tabelit.

Table of abbreviations and terms

ACID	Atomicity, Consistency, Isolation, Durability.
AHP	Analytic Hierarchy Process
ANSI	American National Standards Institute
AS^{3}AP	ANSI SQL Standard Scalable and Portable
API	Application Programming Interface
BSON	Binary JSON
CIO	Chief Information Officer
CLI	Command Line Interface
CPU	Central Processor Unit
CSV	Comma-separated values
DBA	Database Administrator
DDL	Data Definition Language
DML	Data Manipulation Language
FAHP	Fuzzy Analytical Hierarchy Process
GPL	General Public License
GUI	Graphical User Interface
JSON	JavaScript Object Notation
HW	Hardware
LDAP	Lightweight Directory Access Protocol
MVCC	Multiversion Concurrency Control
SME	Small and medium sized enterprises
SQL	Structured Query Language
DBA	Database Administrator
DBMS	Database Management System
PL	Procedural Language
ODBC	Open Database Connectivity
OLTP	Online Transaction Processing
OS	Operating System
OSDB	Open Source Database Benchmark
RDBMS	Relational Database Management System
RPM	Red Hat Package Manager
SMP	Symmetric Multiprocessing

SSL	Secure Sockets Layer
SSH	Secure Shell
SW	Software
TPC	Transaction Processing Performance Council
UDF	User Defined Function
WAL	Write-Ahead Logging
XML	Extensible Markup Language

Table of Contents

List of figures

List of tables

1. Objectives of database selection

1.1. Small and medium-sized enterprises in Estonia

Small and medium-sized enterprise (SME) is a company with number of employees less than 250 [1]. The turnover should not exceed 50 million euros for medium sized enterprise (up to 250 employees), 10 million euros for small enterprise (up to 50 employees) and 2 million euros for so-called micro-enterprise (up to 10 employees) [1]. In 2013 the percentage of these enterprises was up to 99,8% both in EU and Estonia providing two-thirds of jobs and more than 50% of value added produced by businesses [2].

SMEs are reliable basis of Estonian economy with 78% of jobs and 74% of value added [2]. In fact only 6,8 % (Figure 1) of Estonian SMEs are medium-sized enterprises [3] .

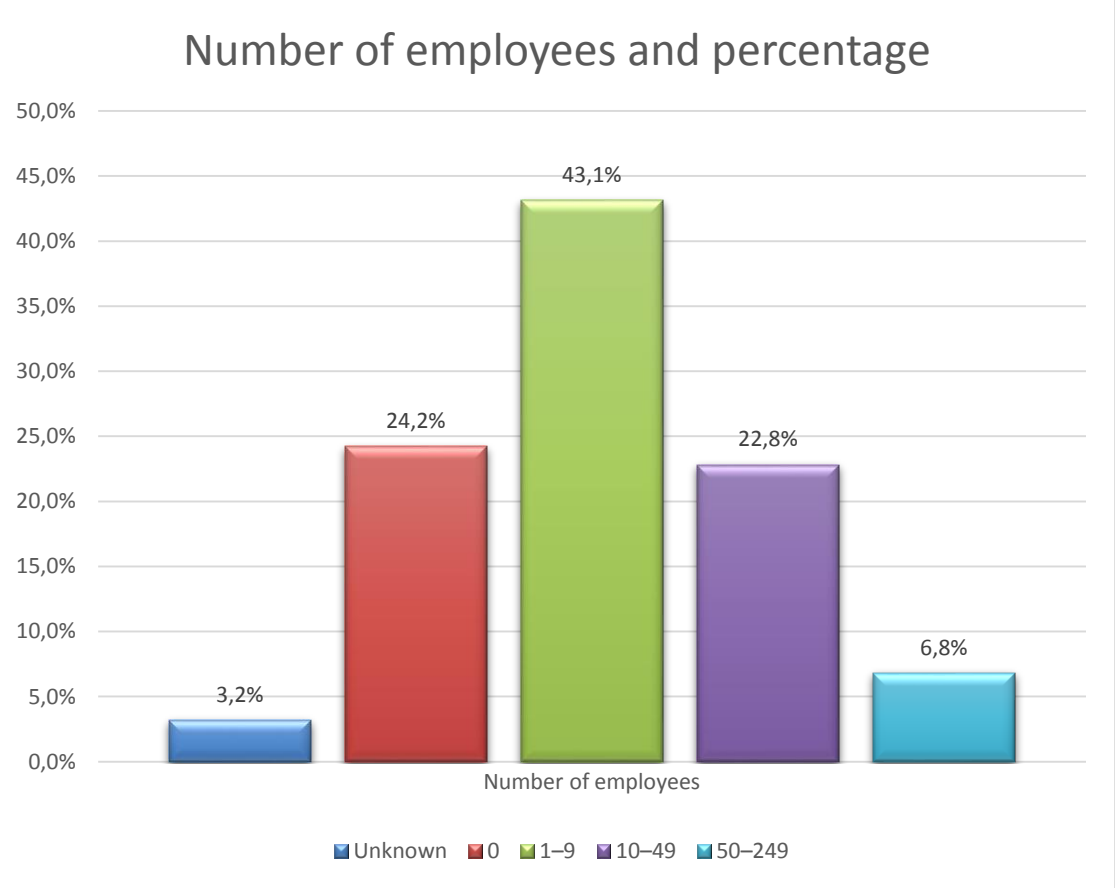

Figure 1. Number of employees and percentage

The business areas of Estonian SMEs in 2012 represented on Fig.2 depict almost homogeneously distributed graph with manufacturing area at the top [3]. This situation differs from European average where wholesale and retail trading, manufacturing, services and construction are leading [4]. Unusual small value belongs to the retailing can be based by the fact that this business area in Estonia occupied mainly by large (for Estonian market) retailers with a larger number of employed personal more than 249.

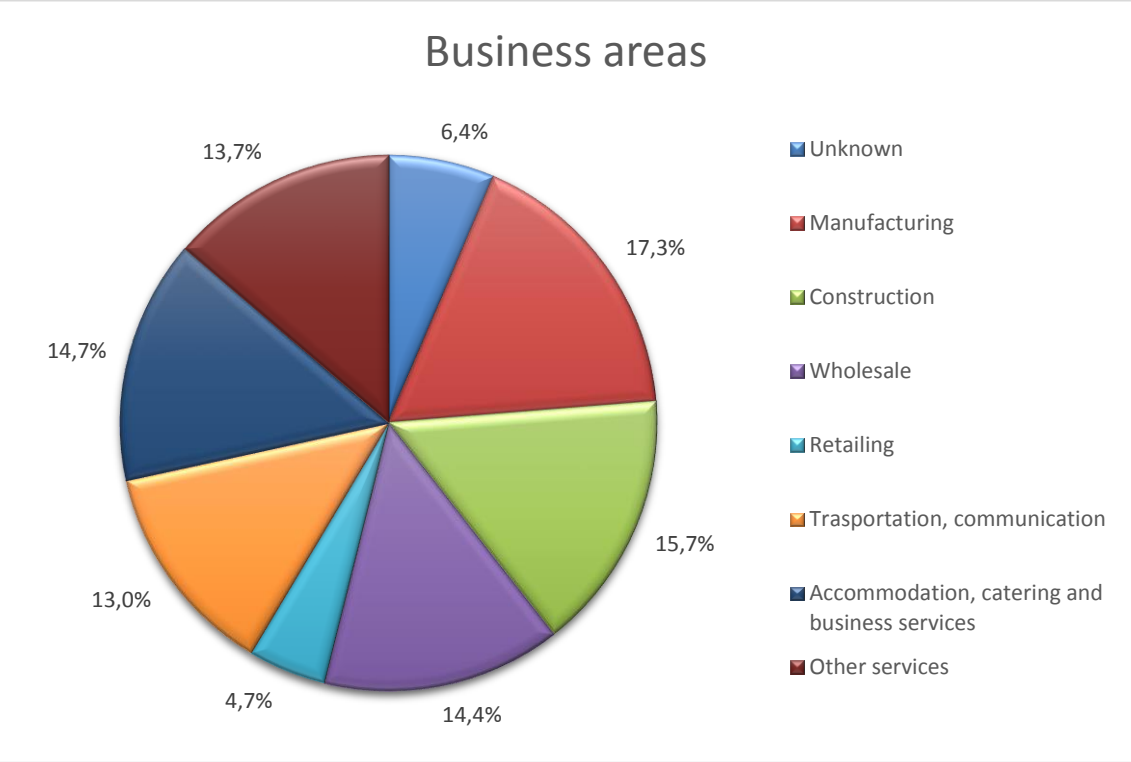

Figure 2. Business areas of Estonian SMEs in 2012Business areas of Estonian SMEs in 2012

According to Statistics Estonia e-commerce companies or in other words online shops have significant growth during the last period [4]: if in 2001 there were only 35 e-commerce companies on Estonian market, then already in 2011 there were about 340 such companies on the market. However, if in 2001 average number of employees was between 6 and 9 so in 2011 this number was 2 employees per company. Additionally, although the companies had revenue about 70 million euros in 2011 this number is only 1.5% of overall income of retailing area. So in fact, this business belonging to SME group of companies but has no crucial role on the Estonian market.

1.2. Database Management Systems

DBMS is software application designed to administrate and maintain databases providing mechanism of interactions between clients and database storage. The client can be for example DBA that uses DDL or GUI for database administration. As well, the client can be client application (web or desktop) that uses application interface, for instance ODBC, to interact with data or an analyst that using DML queries collects necessary data. DBMS facilitates the management of data stored in a database and its logical structure.

History of today's DBMSs starts in 1970s when E.Codd published his work concerning relational model of data presentation. This time Codd worked for IBM Research Laboratory where basing on his document was developed System R. This DBMS demonstrated structured query language (SQL), advantages of relational model and served as example for a number of commercial DBMSs [89].

The relational model brings the meaning of relation and its attributes. In the RDBMS, the concept of relation is represented on the physical level as database table and relation attributes as table columns. The relational model provides data integrity by implementation of constraints. E. Codd was the first who noticed that data redundancy in relations have to be avoided, he proved that the redundancy can lead to inconsistency of data and proposed the procedure that called normalization [89]. DDL is a concept of a language that is able to describe database structure and DML allows manipulating the data. DMLs can be divided into procedural (what data and how manipulate it) and non-procedural that concentrated only on what data need to be manipulated. SQL belongs to non-procedural family.

Databases have to provide the ability multiple users to interact with data. The essential notion of concurrency control is transaction. This is undivided set of actions that can be executed completely or in case of failure completely cancelled. For example, in the database of the bank system in the scope of one transaction money should be withdrawn from one account and placed to another. This operation can be done in full range or fully cancelled, otherwise this money is able to disappear in case of some system failure. The transaction realizes ACID principle: atomicity, consistence, isolation and durability [89]. T.Haerder and A.Reuter in 1983 proposed this abbreviation to describe features that transactional database should guarantee. Under the notion of atomicity is already mentioned indivisibility of transaction, the consistence means that data in database have

to be consistent, isolation requires independent execution of transactions and durability that results of committed transactions cannot be lost [89]. Online Transaction Processing (OLTP) is behavioral data model of DBMS typically concentrated into the online transactional data operations with multiple connected users. In OLTP systems users execute simple repeating but transactional queries and capability to perform large number of simultaneous isolated transactions plays most significant role in these systems. Contrary to OLTP, in Online Analytical Processing (OLAP) systems queries may have unique pattern, the queries are much longer, they are more complicated and may be non-transactional. In these systems transactional performance is less important. OLAP systems are usually used in banking sector, data warehouses, in large companies by analysts while OLTP model is popular in banking sector as well for client operations, manufacturing, sales, bookkeeping, Client Relations Management (CRM), Enterprise Resource Planning (ERP).

Data integrity and consistence mechanisms of non-relational or sometimes called NoSQL DBMS are based on different from relational model principles. There are key-value stores, XML, JSON or BSON based stores, object, graphs and document databases. These DBMS use different data models and serve for number of different purposes. They can be very effective in some areas, for instance document-oriented DBMS is able to be implemented in data warehouse [89].

1.3. Common selection criteria of DBMS

The primary goal of this selection to figure out what DBMS is most optimal for particular purposes of the system, taking into account performance and reliability as necessary components of successful everyday operation. Typically, investments are suggested in the framework of a project are limited by current performance requirements. However, today's selection of DBMS have to be done with an eye to the possible future growth of the system. This fact leads to understanding the next substantial ability of DBMS – the scalability. Usually, under this definition is meant the ability of the system to adopt a growth of working load or in case of DBMS growing amount of client queries. The planning of the issue how the system processes data in case of essential increment of the load need to be done in the very beginning. No doubt, the simplest way is to change the hardware and to buy new more powerful server. This approach is named scaling up or vertical scaling. However, large companies cannot afford to use vertical scaling in their

million queries per minute business systems. They have to scale horizontally that mean to use multiple instances of the DBMS named as nodes to split and balance the load. For example, some forum application can be divided to functional partitions like “forum” and “users”. The “forum” node can hold the data about forum messages and the “users” part stores the users’ data. This technique is often called data sharding or partitioning. But if DBMS selected as the data processing unit of the system doesn’t support technics of horizontal scaling or this support is somehow limited it can ruin the system in the future. For instance, developers of car selling site had added a new function that allows to find car insurance history by its registration number. This function became extremely popular and as the result, the usage of database engine drastically increased. Nevertheless, the DBMS selected for this project does not support horizontal scaling or its implementation requires paying high license fee. As consequence, the result of the implementation of this advanced function is rather negative because many site visitors refused of using this low overloaded site. All said above is the reason why the scalability is one of the important criteria of DBMS selection.

Reliability of DBMS have to be considered as complex parameter that based on backup, restoring, security and transaction durability issues. Reliable OLTP DBMS have to follow ACID principles to ensure reliability of transactions as well. Unfortunately, under DBMS backup is often mistaken for replication of DBMS. Indeed, replication can be used for backup if it used with time delay on slave node [19], but these two processes serve different purposes. Replication is typically implemented for already mentioned horizontal scaling when reading operation delegated from master server to slave one for load optimization. If data updates are being repeated on the slave immediately, there is no way to rollback if any unwanted data modification has occurred. Backup is rather separate procedure that periodically makes data snapshot. The most common way is making database dump manually. However, there are many possibilities, including third-party utilities, to backup data automatically. Definitely, security issue need to be concerned here. It is necessary to mention that security measures as integral part of DBMS reliability guarantee that data remains valid and server administration data is not compromised. In the process of DBMS selection have to research what level of security should be provided and whether selected candidate is able to ensure required degree of security. Modern DBMSs are supplied with multilevel security mechanism and realization of this mechanism is different. Some DBMSs propose using of complicated

role-based mechanism like Oracle Database or Microsoft SQL server that essentially simplifies group security management and others do not support this mechanism. Some DBMSs are able to encrypt data and work with encrypted connection like SSH. Multiversion concurrency control is the mechanism that as part of DBMS reliability ensures accordance with ACID principles of transactions to act and different DBMSs have different implementation of this mechanism. For example, MyISAM, first storage engine of MySQL, does not support transactions, so it cannot be used in OLTP model. All this issues need to be examined during DBMS selection process from reliability point of view.

When professionals speak about functionality of DBMS, they usually mean entire spectrum of its abilities like available APIs, stored procedures and triggers, user-defined data types and functions, search engines. All these features can play crucial role in processes of development, operation and maintenance of the system. A lack of some part of functionality can stop normal execution of these processes making it unsuitable for performing system tasks. For instance, if some function is not realized in the API developers have to find their own way to add required functionality to the project that requires extra time and as consequence money. Moreover, this devious way sometimes not enough optimal that makes constructed system more complex and as result less fault tolerant. Thus, supported functionality is important feature of DBMS.

Mobility of DBMS or platform-independence is also important if circumstances force to migrate to another software or hardware platform. Some of the DBMSs are limited with certain platform and most vivid example that can be cited here is Microsoft SQL Server that is able to run only with Microsoft Windows family. Not only software but also hardware platform can engage some limitations like unsupported type processors or multiprocessors. In addition, mobility of DBMS is the measure how easy and quickly DBMS can be deployed on new platform.

Data model is probably the first thing that should be decided in the very beginning of every project where intended to use DBMS. What type of data model whether traditional relational model or trendy NoSQL model in this a lot depends on developers' skills and project tasks. For some applications, NoSQL model can bring significant breakthrough of productivity. Nevertheless, there are not too many skilled developers and CIOs prefer

to have a deal with well-known technology. So all these aspects are also should be taken into account immediately before starting the procedure of DBMS selection.

Mentioned above performance is generally measured using TPC benchmarks. Transaction Processing Performance Council is non-profit organization dealing with benchmarks of co-simulation different server hardware with commercial DBMS. De-facto the TPC benchmarks became industrial standard for proprietary DBMSs and server equipment. There are several benchmarks (TPC-C, TPC-E etc.). Every benchmark is pointed to figure out different aspects of computing performance. TPC proposed to use Transaction-per-cent unit to estimate efficiency of investments into the certain platform. There is also new TPC-Energy specification with Energy Metrics created to find out optimal conjunction of price, performance and energy requirements. The results of benchmarks are being regularly published on the TPC official site and interested persons can acquaint with them. In other words, in case of commercial platform performance can be approximately estimated using TPC benchmark results. In the meantime, there are disadvantages in TPC benchmarks. First, benchmarks concern the definite composition of hardware and software platforms. Hardware is usually produced by major server hardware manufacturers who are members and sponsor of TPC project. It means hardware made by smaller manufacturers like Ordi cannot be found from the list. Same situation is on the software side – all tested DBMSs belong to product lines of well-known commercial DBMSs from such giants of the industry like Oracle, IBM and Microsoft. Open source DBMS are not tested in the TPC benchmarks. Second, all these benchmarks are too general and results achieved in the TPC benchmarks do not reflect performance of a custom application. As a rule, a performance demanding application needs in their own benchmarks based on task of the application. Analytic tools and third-party utilities are also useful in this situation. Some of the analytic mechanisms are included in the DBMS installation packet. There is slow queries log in MySQL that can be used for their tracking. Maatkit or its successor Percona Toolkit are sets of third-party utilities that can help to optimize slow queries. It is clear that low performance of DBMS affects negatively the selection.

1.4. Important criteria from SME point of view

Apparently, the cost of DBMS license can be most important selection criteria for Estonian SMEs. When procedure of selection has just started from blank and there is no

legacy software that is able to work with some particular DBMS, it is very reasonable to weigh the possibility to implement an open source DBMS in the enterprise ecosystem. Very often small enterprise cannot afford to pay thousands of euros only for DBMS license. Today most part of work specific applications like bookkeeping, warehouse or monitoring systems have open source analogs without conceding functionality. Linux OS became a de facto standard in the webhosting business and even large corporations like Facebook and Google are using composition of this OS with open source DBMS. There is only reason to prefer proprietary DBMS is the existence of an application that is limited by one DBMS type and the application is tightly connected to the main working. The relevant example of such kind of application is very popular in Estonia is SAF bookkeeping software that is able to use the only one Microsoft SQL Server as database engine. In other cases, usage of open source DBMS is justified. Here arises the important issue of DBMS mobility. Platform independence and ability quickly be switched to another platform is definitely important for SME. Three of most popular open source DBMSs – MySQL, PostgreSQL and Firebird SQL server are able to run on several platform including most popular Windows, Linux, OSX and BSD. If a company has renewed its server park or acquired new software the ability to migrate to new platform can be crucial. For instance, the company selected new DBMS can be installed on the existing server with Microsoft Server OS, but it is decided to buy new server with Linux after certain period. The migration in this case can be performed painless because on the company can use same DBMS on the new platform. As already mentioned before the inner structure, supported SQL syntax and data types are different for different DBMSs and transition process from one DBMS to another can be very complicated process. In case of DBMS that is compliant with different OS platform switching is easy routine process. Thus, issue of DBMS mobility for SME have to be considered as one of the most essential criteria.

Besides, it have to be here mentioned that SME needs its own dedicated DBMS that located directly in the physical domain of SME, not somewhere in a provider cloud. Only thus, everyday working activity of the enterprise will not be suspended in case of internet connection failure. Nevertheless, effective energy consumption of server equipment is power efficiency of all its components including, inter alia, DBMS. The energy efficiency of DBMS can be evaluated as number of transactions performed per one consumed watt of energy. Although, power efficiency cannot be most crucial factor for SME yet it have

to be considered even for this type of enterprises. This suggestion is based on the fact that middle-sized webhosting provider belongs to SME category and for this enterprises the difference in DBMS energy consumption can be perceptible. There is high probability that middle-sized webhosting provider is interested in scaling out of DBMS. Consequently, the DBMS must be scalable that is why scalability of this DBMS have to be verified as well.

Functionality as measure of what and how DBMS is able to do should be thoroughly investigated during DBMS selection process for all types of enterprises, not only for SMEs. This investigation ensures that DBMS is valid for application tasks and as result can be selected. The basic features like triggers or stored procedures are obviously supported by majority of DBMS, but a lack of some part of functionality can be met for certain tasks.

For open source DBMS is substantial issue the degree of DBMS information availability or in other words, whether DBMS documentation is detailed enough or not. It is very important when every moment of implementation or every member of API described by specialists in details. Popularity among developers and sequentially large community, which can help in difficult circumstances, is definitely positive sign to select DBMS.

After the revision of business areas (Figure 2) it is possible to conclude that most of computing systems of SME belongs to OLTP class. Moreover, performance for such systems can be evaluated as value of transactions processed in second. For some application areas, where todays Estonian SMEs are presented, DBMS performance have to be taken into account. Otherwise low performance value can ruin the business. For example if the slow DBMS is a part of CRM software. This mismatch can lead to missed orders, because customers cannot wait until their order pass through the system and next time can order the item from somewhere else.

Availability of software tools for DBSM administration, monitoring and optimization is also important for SMEs. These tools allow the DBA to save a lot of time. Already mentioned Maatkit [88] is a set of 31 GPL licensed utilities (Perl scripts) that are able to simplify significantly processes of monitoring, backup, replication and so on of MySQL DBMS family. This fact can make MySQL very attractive for selection and allows saving

money of SME that possibly cannot afford to invest money in such kind of proprietary tools.

In addition, the DBMS have to ensure proper level of reliability. The data cannot be lost and unauthorized access must be restricted. This is common requirement for SMEs and larger companies. The requirement is because different DBMS have different implementation of security protocols and backup. Consequently, the issue of reliability have to be investigated in the scope of intended tasks.

1.5. The Analytic Hierarchy Process as the basement of evaluation platform.

The Analytic Hierarchy Process is the decision-making methodology developed by mathematician Thomas L. Saaty in 1980s. The technique facilitates the way to find out not the perfect decision for a case, but the decision that is the best between its alternatives. Thomas L. Saaty compared his technique with the measurement without starting zero point based only on the priority hierarchy and pairwise comparison. In other words, the priority hierarchy plays the role of system of axis that decision maker should build up first guided by the goal of the case. The pairwise comparison is the process to achieve the goal. This process composes the consistent reciprocal matrix from elements produced from comparison of measured values.

Suppose there is a collection of objects S_1, S_2 to S_n. As result collection of judgments on these objects pairs (S_i, S_j) can be expressed by $n \times n$ matrix A (1) [5].

$$A = (a_{ij}),\ where\ i, j = 1 \ldots n \qquad (1)$$

For all entries a_{ij} are valid next rules. First, if $a_{ij} = \alpha$ then $a_{ji} = 1/\alpha$ and $\alpha \neq 0$. Second, the matrix is reciprocal what means always $a_{ii} = 1$ and if S_i and S_j have same weight both a_{ij} and a_{ji} equal1.

Thus, the matrix of judgments can be presented as (2).

$$A = \begin{pmatrix} 1 & a_{12} & . & a_{1n} \\ 1/a_{12} & 1 & . & a_{2n} \\ . & . & . & . \\ 1/a_{1n} & 1/a_{2n} & . & 1 \end{pmatrix} \quad (2)$$

A matrix of n order needs to perform $n(n-1)/2$ judgments [5].

Assume that there some number of physical items $S_1, S_2, \ldots S_n$ that can be weighed with a certain degree of accuracy, so ω_1 is the weight of S_1 and ω_2 is the weight of S_2 and so on. If S_1 weights 256 grams and S_2 weights 128 grams one can judge that S_1, is 2 times heavier than S_2 , and the judgment of pairwise S_1, and S_2 can be mathematically expressed as equitation (3).

$$a_{12} = \frac{\omega 1}{\omega 2}, \quad (3)$$

Thus, judgments can be described by equitation (4).

$$a_{ij} = \frac{\omega i}{\omega j} \text{ , where } 1 \leq i, j \leq n \quad (4)$$

Consequently, the matrix of judgments is the next (5).

$$A = \begin{pmatrix} \omega_1/\omega_1 & \omega_1/\omega_2 & . & \omega_1/\omega_n \\ \omega_2/\omega_1 & \omega_2/\omega_2 & . & \omega_2/\omega_n \\ . & . & . & . \\ \omega_n/\omega_1 & \omega_n/\omega_2 & . & \omega_n/\omega_n \end{pmatrix} \quad (5)$$

Nevertheless, if one needs to compare non-physical values like subjective judgments of people about some property of an object or it is impossible to measure the weight exactly, in this case finding of the judgment can be complicated or even non-executable task.

It is clear that such kind of judgments' deviation have to be mathematically limited. The elements of row *i* of cited above judgment matrix are $a_{i1}, a_{i2}, \ldots a_{ji}, \ldots, a_{in}$. According to equitation (4) all elements a_{ji} are relations of values like ω_i/ω_1 and ω_n/ω_n. So one multiply first element a_{i1} by ω_1, second a_{i2} by ω_2 and so on up to the last one (6).

$$\omega_i = \omega_j \frac{\omega i}{\omega j} \text{ , where } 1 \leq i, j \leq n \qquad (6)$$

The result of this operation is a row contains of ω_i, meantime in common case there is a statistical dispersion of values around ω_i [5]. As consequence, the value of ω_i can be calculated as average value of the row. That means basing on this assertion and equitation (4) ω_i can be described by equitation (7).

$$\omega_i = \frac{1}{n} \sum_{j=1}^{n} a_{ij}\, \omega_j \text{ , where } 1 \leq i, j \leq n \qquad (7)$$

However, if all elements of equitation (7) a_{ij} and ω_1 are available it is necessary to determine whether this equitation has the only solution or not. Deviation of a_{ij} depends on accuracy the estimation of ω_i/ω_j. The more exact calculated ω_i/ω_j the more close a_{ij} to real magnitudes. Thomas L. Saaty proposed that instead of rank of the matrix *n* another feature of square matrix should be used to find the solution. This is maximum eigenvalue of the matrix λ_{max}.

$$\omega_i = \frac{1}{\lambda_{max}} \sum_{j=1}^{n} a_{ij}\, \omega_j \text{ , where } 1 \leq i, j \leq n \qquad (8)$$

In fact, the comparison matrix is symmetrical by definition because all elements of the matrix's trace equal to one. That means all eigenvalues are real. The calculation of maximum eigenvalue is well-known mathematical problem and will be demonstrated further in this work with real magnitudes. Generally, deviations of a_{ij} is able to cause large deviations in ω_i and λ_{max} , but not in case of reciprocal matrix. As result, the equitation (8) has the solution.

The goal of the described above method is to get priority vectors. These vectors are principal eigenvectors of the comparison matrix and can be calculated by normalization of matrix's entries [5]. The sum of eigenvector's elements after normalization is 1 and magnitudes represent weight of priority for one element ω_i. These priorities describe the

importance of ω_i in the scale to other weights $\omega_{1...}\omega_n$. The greater number means higher position in overall ranking of compared items.

The consistency index of matrix (CI) is proposed to calculate using maximum eigenvalue of the matrix λ_{max} and rank of the matrix n (9). This index is the measure of deviation of a matrix from consistency.

$$CI = \frac{\lambda_{max-n}}{n-1} \quad (9)$$

In National Laboratory of Oak Ridge were computed random index of consistency for matrices with ranks up to 15. The results limited with rank 10 are presented below (Table 1) [5]. The first (Table 1) presents is the rank of a matrix and second is its random index.

Table 1. Random indices

1	2	3	4	5	6	7	8	9	10
0	0	0.52	0.89	1.11	1.25	1.35	1.40	1.45	1.49

The consistency ratio of a matrix (CR) is relation between consistency index CI and relevant random index RI.

$$CR = \frac{CI}{RI(n)} \quad (10)$$

If CR of a matrix does not exceed value 0.1, the matrix can be named consistent.

There are several methods described below that can used for normalization of matrixes. The first method is to sum all entries in the rows, than to get total sum the row sums and divide every row sum by total sum. The results are priorities of the objects [87]. The next method proposes to sum the entries of the columns, than it is necessary to divide the values of the entries by the sum and the last step is calculation the average value for every row [87]. These values are the priorities. This method is used in this thesis for further normalization of the comparison matrixes. The alternative method [15] supposes to square the matrix (multiply it by itself) and then to divide the row sums by their total sum.

2. Proposed methodology of evaluation.

As was previously described in Chapter 1.4 the most valuable criteria required to estimate DBMS selection are availability of management tools, thoroughness of technical documentation and community support, mobility as measure of platform independence, functionality, reliability, scalability, performance and power efficiency. Some of the listed DBMS features like performance or power efficiency can be measured in real values, but the rest features can be only subjectively estimated. Appropriately, the proposed methodology should be able to assess combination of physical values, like seconds and watts, and judgments about intangibles. The AHP methodology is versatile self-complete mathematical mechanism suitable to provide relative measurement of intangibles [5]. The methodology found its application in various areas of business and science – psychology, real estate, military, minerals mining and education. This is the reason to apply the AHP as the basis of the methodology of DBMS evaluation.

2.1. AHP based criteria assessment.

In the beginning of the pairwise comparison process, it is necessary to define DBMS parameters that should important for Estonian SME. These criteria were already described and proposed as the most important for the DBMS (Table 2) in Chapter 1.4.

Table 2. Examined criteria and their abbreviations.

Abbreviation	**Criterion**
Tools	Available administrative and monitoring tools
Docs	Documentation and community support
Mob	Mobility
Rel	Reliability
Fun	Functionality
Scl	Scalability
Perf	Performance

Abbreviation	Criterion
Power	Power efficiency

This comparison of intangibles requires relative values that characterize the dominance of one parameter over another. The Fundamental Scale (Table 3) proposed by Thomas L. Saaty [5] contains the required values.

Table 3. The Fundamental Scale of absolute numbers

Intensity of Importance	Definition	Explanation
1	Equal importance	Both criterion are equally important
2	Weak of slight	
3	Moderate importance	Experience or judgment slightly favor one criterion over another
4	Moderate plus	
5	Strong importance	Experience or judgment slightly favor one criterion over another
6	Strong plus	
7	Very strong or demonstrated importance	A criterion is favored very strongly over another; its dominance is demonstrated in practice
8	Very, very strong	
9	Extreme importance	The evidence favoring one activity over another is of the

Intensity of Importance	Definition	Explanation
		highest possible order of affirmation

Thus, having the values of relative importance it is possible to compose the pairwise comparison table (Table 3). The table used abbreviations described in Table 2.

Table 4. The Comparison Matrix of selection criteria.

	Tools	Docs	Mob	Rel	Fun	Scl	Perf	Power
Tools	1	1	1/7	1/7	1/7	1/3	1/5	1/3
Docs	1	1	1/7	1/7	1/7	1/3	1/5	1/3
Mob	7	7	1	1	1	5	3	5
Rel	7	7	1	1	1	5	3	5
Fun	7	7	1	1	1	5	3	5
Scl	3	3	1/5	1/5	1/5	1	1/3	1
Perf	5	5	1/3	1/3	1/3	3	1	3
Power	3	3	1/5	1/5	1/5	1	1/3	1

Table 5. The Comparison Matrix of selection criteria in decimals.

	Tools	Docs	Mob	Rel	Fun	Scl	Perf	Power
Tools	1.00	1.00	0.14	0.14	0.14	0.33	0.20	0.33
Docs	1.00	1.00	0.14	0.14	0.14	0.33	0.20	0.33
Mob	7.00	7.00	1.00	1.00	1.00	5.00	3.00	5.00
Rel	7.00	7.00	1.00	1.00	1.00	5.00	3.00	5.00

Fun	7.00	7.00	1.00	1.00	1.00	5.00	3.00	5.00
Scl	3.00	3.00	0.20	0.20	0.20	1.00	0.33	1.00
Perf	5.00	5.00	0.33	0.33	0.33	3.00	1.00	3.00
Power	3.00	3.00	0.20	0.20	0.20	1.00	0.33	1.00

Using Xcas mathematician software kit the maximum eigenvalue of the comparison matrix (Table 4) λ_{max} has been calculated – it equals to 8.22. According to cited above equitation (9), the consistency ratio has been calculated and compared with 0.10 (11).

$$CR = \frac{CI}{RI(n)} = \frac{(8.22-8)/7}{1.41} = 0.03143 \leq 0.10 \qquad (11)$$

Consequently, the comparison matrix is consistent.

Nevertheless, in order to increase the accuracy of estimates the table (Table 4) have to be normalized. Here the sums of values in columns: 34.00, 34.00, 4.02, 4.02, 4.02, 20.67, 11.07, 20.67. The normalized matrix has been produced by dividing each value in columns by corresponding sum of columns (Table 6).

Table 6. The Normalized Comparison Matrix

	Tools	**Docs**	**Mob**	**Rel**	**Fun**	**Scl**	**Perf**	**Power**
Tools	0.03	0.03	0.04	0.04	0.04	0.02	0.02	0.02
Docs	0.03	0.03	0.04	0.04	0.04	0.02	0.02	0.02
Mob	0.21	0.21	0.25	0.25	0.25	0.24	0.27	0.24
Rel	0.21	0.21	0.25	0.25	0.25	0.24	0.27	0.24
Fun	0.21	0.21	0.25	0.25	0.25	0.24	0.27	0.24
Scl	0.09	0.09	0.05	0.05	0.05	0.05	0.03	0.05
Perf	0.15	0.15	0.08	0.08	0.08	0.15	0.09	0.15

Power	0.09	0.09	0.05	0.05	0.05	0.05	0.03	0.05

The goal of the comparison and optimization process is to get vector of priorities that represents distribution of DBMS parameters' priorities. The vector has been calculated (Table 7) by summing of values in rows and diving this sum by rank of matrix (eight in our case).

Table 7. The vector of priorities.

Criterion	**Priority weight**
Available management tools	0.03
Documentation and community support	0.03
Mobility	0.24
Reliability	0.24
Functionality	0.24
Scalability	0.06
Performance	0.12
Power efficiency	0.06

The higher value of priority weight means the higher importance of the criterion. According to this three most important criteria are mobility, reliability and functionality with priority weight equals to 0.24. On the second place of priority is performance with priority weight 0.12, the third place is divided between scalability and power efficiency priority weight 0.06. On the last place are criteria "Available administrative and monitoring tools" and "Documentation and community support" with value 0.03.

Now the priority vector can be used to investigate different DBMSs in order to evaluate their suitability for Estonian SMEs. A number of DBMSs can be selected for evaluation using very similar comparison matrixes for every proposed criterion. Then results can be

multiplied by priority weights and summed in scope of one particular DBMS. The DBMS that collects the maximum number of points is most suitable for proposed criteria.

2.2. Method for evaluation DBMS performance.

Instead of mentioned above TPC benchmarks, the open methodology based on the ANSI AS^{3}AP standard is proposed to be used for performance evaluation.

The history of AS^{3}AP began in the end of 1980s when Carolyn Turbyfill (Sun Microsystems), Cyril Orji (University of Illinois at Chicago) and Dina Bitton (DB Software Corporation) started their work on a new benchmark for database performance evaluation. Before this, the widespread Wisconsin Benchmark was considered as common milestone in deal of comparison for a number of different systems. The Mentioned above authors found that the database and queries of the Wisconsin Benchmark after instantiation were not able to provide adequate framework for comparative benchmarking [6]. In this work the authors proposed new model that they named as ANSI SQL Standard Scalable and Portable (AS^{3}AP). The described system have to be compliant and automated set of test that is able to cover all aspects of evaluation of database processing power. Moreover, the system have to be system independent, scalable and provide its results in same way for all range of tested systems. The proposed benchmark system includes database with certain schema that contains five tables described in Table 7 [6].

Table 8. AS^{3}AP database tables

Table name	**Characteristics**
uniques	Contains only unique values.
hundred	Most of the columns have exactly 100 unique values.
tenpct	Most of the columns have 10% unique values.
updates	For three different types of indices.
tiny	A one column and one row table

All those tables use same data types with the same names. The data types proposed to use in the tables are most commonly used data types likes both types of integers (signed and unsigned), exact decimal and floating point, strings with variable and fixed length and 8 character. Furthermore, it is the authors supposed that the database table could be queried by application that they called benchmark that can be one large or set of smaller programs [6]. The single-user benchmark module tests selection, joining, projection, aggregation and updating queries and utilities for data loading and database structuring [6]. The goal of this module is to check compatibility with basic required DBMS functionality described in ANSI SQL 2 Standard [6].

The AS^3AP based benchmark suit can be used in evaluation of DBMS data processing performance criteria for OLTP load model. Such benchmark set is the Open Source Database Benchmark (OSDB) project that is successor of one of the Compaq Computer Corporation projects [7] and based on AS^3AP benchmark. The OSDB is published under GNU General Public License version 2.0 (GPLv2) .The OSDB project is intended to be database and system independent benchmark facilitates performance analyzing of different DBMS using same set of SQL queries. This goal was achieved by separation of database access mechanism and benchmark tests. The project is written mainly in C++ that simplifies significantly usage of native C libraries of DBMS-s for development purposes. The first version of the project accessible for download version 0.4 was published on January 20 2001. The version used in this work (version 0.90) was uploaded on 22 December 2010. This version contains code for accessing and testing MySQL DBMS family. The architecture of the OSDB application is presented in UML class diagram below (Figure 3).

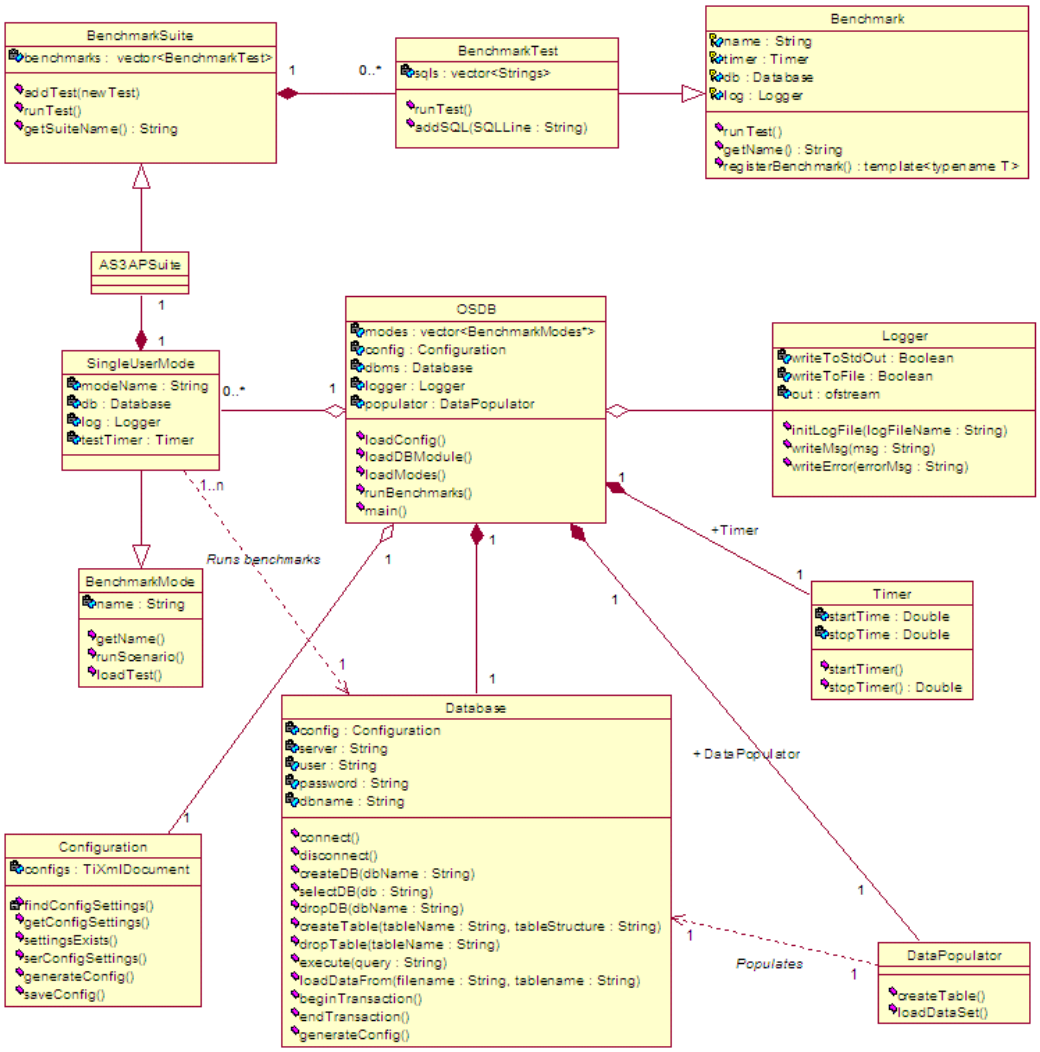

Figure 3. OSDB project class diagram.

On the top of the application hierarchy is OSDB class that instantiates main supplementary classes Timer, Database, DataPopulator and array of the objects belonging to BenchmarkMode class. The instance of Timer class is used to measure time elapsed for the benchmarks. The purpose of the Database class is to create an interface to one of the DBMS that encapsulates DBMS specific interaction mechanism. For instance, the Database class creates the connection to DBMS, then the database with name "osdb" in the DBMS and five tables further data operations. The DataPopulator is the class that gets Database class object by reference and populates it with data from five files located in the file system. This data-loading feature is implemented in the Database class and the implementation is distinct in different DBMS-s. The BenchmarkMode is an abstract class used to instantiate objects of SingleUserMode wrapper class that is responsible for

running benchmarks by using instance of AS3APSuite class. The benchmarks are set of classes to hold SQL query string; every class contains the only string. The AS3APSuite class loads the query strings and run the tests.

The database structure used in the OSDB project is based on AS^{3}AP recommendations [6]. It consists of five tables with already known names uniques, hundred, tenpct, updates and tiny.

Table 9. OSDB database used data types and names.

Nr.	**Name**	**Type**
1	col_key	int(11)
2	col_int	int(11)
3	col_ signed	int(11)
4	col_float	float
5	col_double	float
6	col_decim	decimal(18,2)
7	col_date	char(20)
8	col_code	char(10)
9	col_name	char(20)
10	col_address	varchar(80)

The tables were filled with data from CSV files. The tables have the next numbers of rows:

Table 10. Tables and their numbers of rows

Nr.	**Table name**	**Number of rows**
1	hundred	100902
2	Tenpct	104809
3	Tiny	1
4	Uniques	86995
5	Updates	113893

The listing of used OSDB benchmark queries resides in Appendix1.

2.3. Method for evaluation the power efficiency of DBMS

For this type of evaluation, it is proposed to use power quality analyzer KYORITSU KEW 6310 (Figure 4) to measure consumed power. The analyzer is able to lead accumulative measurement of the power consumed within certain interim and store the measurement result to SD memory card.

To evaluate the power efficiency it is necessary to run benchmark with every of three DBMS and simultaneously to record power consumption. The test have to count the number of performed transactions. It is suggested to that unit for the evaluation can be named as Transaction-per-Watt and overall value can be calculate by equation (2).

$$\mu = \frac{N\ tr}{tPc} \qquad (2)$$

Where μ is the power efficiency, N_{tr} is the number of transactions performed by benchmark with DBMS during certain period of time t in hours and P_c is the consumed power in Wh.

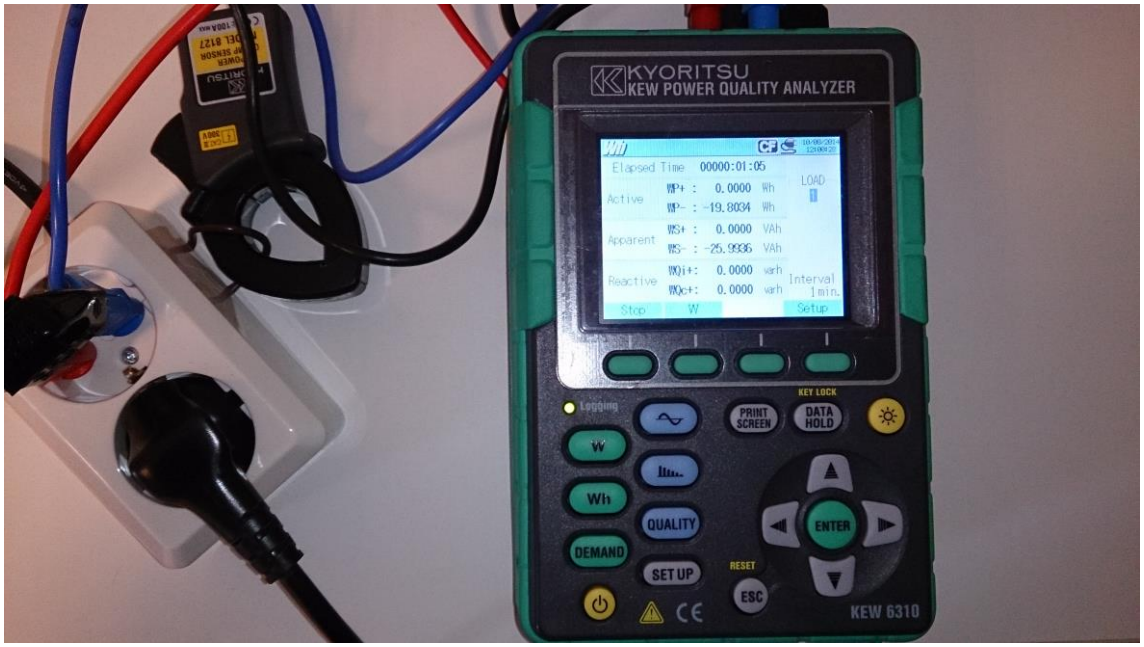

Figure 4. The power analyzer KEW 6310 records power consumption of the testing system.

The power analyzer KYORITSU KEW 6310 was connected using the wiring scheme (Figure 5) supplied by the manufacturer. The correct position of the power clamp sensor (Figure 7) should to be found empirically with the goal to avoid getting negative measured values.

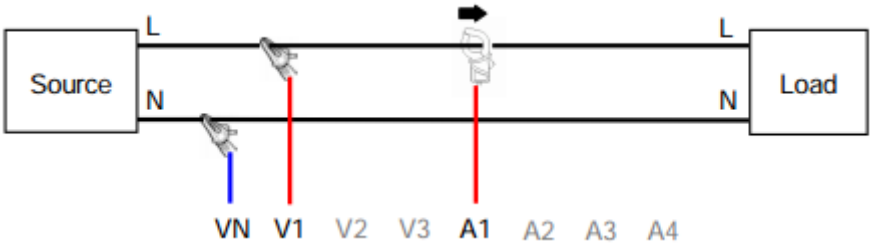

Figure 5. Wiring method for single-phase 2-wire (1ch) of KYORITSU KEW 6310.

Figure 6. The clamp power sensor of KYORITSU KEW 6310.

The power clamp sensor measures the current and the other two inputs are used for voltage measurements (Figure 6). All sensors need to be connected according to manufacturer connection diagram (Figure 8) with input channels defined on Figure 7. The wiring implemented in this work is suitable for single-phase measurements, but using additional input channels it is possible to measure more complicated for example three-phase 3- or 4-wire (with separate neutral) power supplying schemas.

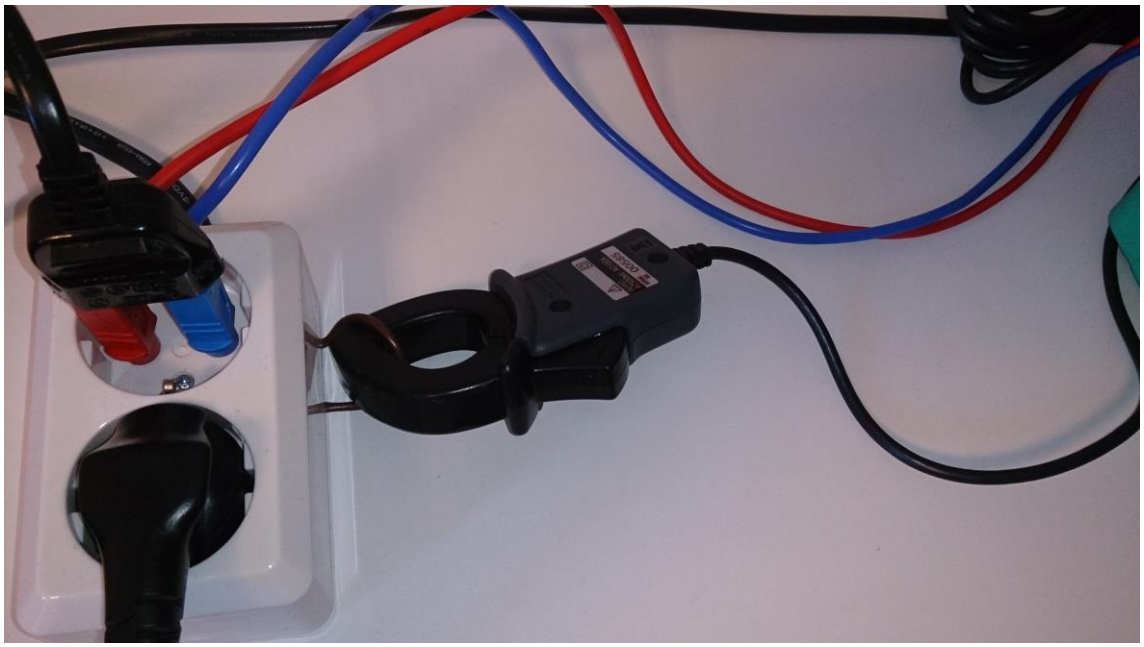

Figure 7. The voltage inputs plugged into the outlet and the current sensor.

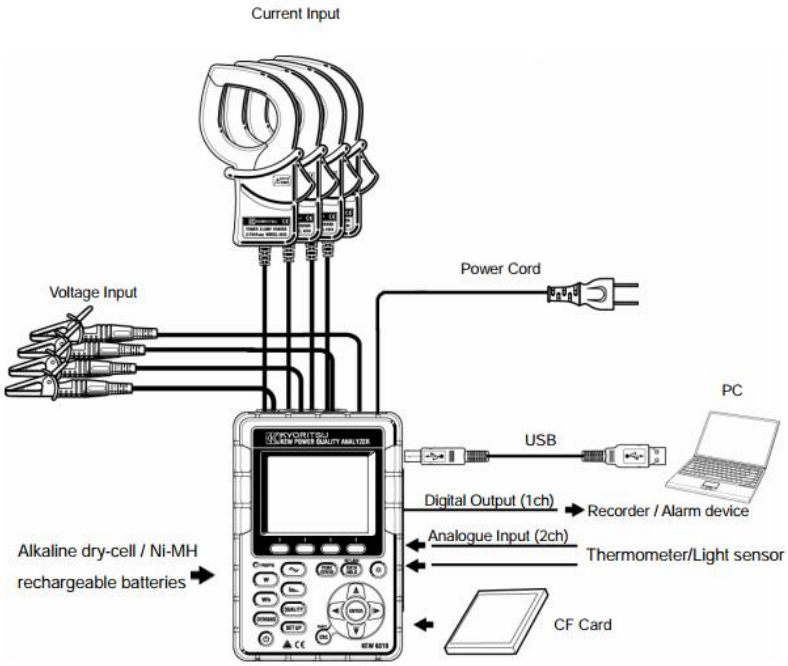

Figure 8. Connection diagram of KYORITSU KEW 6310.

2.4. Evaluation of DBMS

In order to proof the viability of the proposed methodology of the criteria evaluation it is highly necessary to create experimental platform consisting of hardware, OS and installed DBMS software.

For this purpose three most popular open source DBMS – MariaDB (MySQL fork), PostgreSQL and Firebird SQL have been selected to examine and compare using AHP methodology. These three DBMS are open source software, available on most popular server OS platform (Linux, BSD, UNIX and Windows) and support SMP. Linux has been selected as OS of the experimental platform because this the free open source OS is popular on the server market. The architecture of the experimental platform is described in Appendix 2.

2.4.1. Available management tools

All three DBMSs have been examined in the work with a set of CLI and GUI utilities provided by DBMS developers. The task of this chapter to figure out which DBMS is provided most wide set of tools than can used for database optimization and troubleshooting. In the framework of the investigation, the open source tools are preferred therefore as was already mentioned above, a license fee paying can be out of investment focus for Estonian SME, where even hundreds of euros can play crucial role in market competition.

FirebirdSQL server has free of charge multiplatform administration utility FlameRobin tool [59]. The utility has user-friendly GUI that functionally is very similar with pgAdmin III [60] - the GUI administration tool of PostgreSQL. Both utility are able to create, drop database and its objects (tables, views, triggers, functions, procedures etc.), backup and restore databases. However, the functionality of pgAdmin III is wider because it is able to generate server statistics reports, edit configuration files (postgresql.conf, pg_hba.conf, .pgpass). MariaDB is provided with phpMyAdmin [61] web-based administration utility that can be implemented for all described above tasks and can be compared with them and phpPgAdmin[62] - similar web-based administration utility for PostgreSQL. Utility phpMyAdmin allows to monitor database load as well. The lack of all these utilities is inability to be implemented for database analyzing and optimization purposes.

There are several tools available for FirebirdSQL for optimization, monitoring and tracing - IBSurgeon Enterprise Pack [63], FB TraceManager [64], Sinática Monitor [65]. All of them are powerful but commercial software. The only open source tool Easy-IP [66] is limited by Windows platform has same functionality that FlameRobin has. PostgreSQL has impressive list of both open source and proprietary GUI modelling and administration tools [67]. Log scripts analyzer tool pgFouine [68] represents optimization tools developed for PostgreSQL. The tool is able to find out slowly executed queries thereby assisting to eliminate bottlenecks of application performance. PostgreSQL is provided with powerful built-in functionality for optimization and monitoring that are thoroughly documented and freely available [69, 70]. Perhaps this is the reason why number of optimization tools for this DBMS is small. There are companies on the market offering PostgreSQL optimization and consulting services [71, 72], but there is no out-of-box product is able to tune PostgreSQL according to customer needs automatically. MariaDB has the largest list of free tools available for optimization and monitoring. Already mentioned phpMyAdmin displays load in real-time, powerful free innotop CLI utility is able to monitor in up to eleven modes [73]. Percona Toolkit [74] – the successor of mentioned above Maatkit, is versatile set of CLI tools (32 Perl scripts) can be used by DBA for efficient database administration. MySQLTuner [75] and MySQL Tuning Primer [76] are both free scripts that can be implemented for database tuning covering basic performance issues like slow queries, maximum connection number, buffers and cache. From proprietary software, it is necessary to mention popular TOAD [77] and Navicat [78] administration and development GUI tools.

The proposed matrix is cited below (Table 11).

Table 11. The comparison matrix of management tools.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/5	1/3
MariaDB	5	1	3
PostgreSQL	3	1/3	1

2.4.2. Documentation and community

The more completely a system is documented the less time is spent by personnel to find useful information in downtime. As consequence, the less customers remain unsatisfied and products unshipped. The community can play crucial role for technical support of open source DBMSs. To clarify both definitions one can suggest that documentation in this context is technical information provided by main developer team or official site and community is association of independent technical specialists and enthusiasts is able to help to solve a problem by their competent advices.

FirebirdSQL Server is definitely most untypical case from technical documentation style. The DBMS available with four possible types of architectures with three possible dialects of supported SQL has the official site that provides information in the form of unstructured downloadable PDF files at several languages [79]. A search of any information suggests downloading of major part of available materials and attentive research. On the contrary, the documentation of PostgreSQL [80] is well structured by versions and topics and has context search allowing finding required information quickly. It is also possible to download manuals PDF with or without user comments in English, French and Japanese. Both PostgreSQL and FirebirdSQL official sites have FAQ section. MariaDB documentation [81], it less structured than PostgreSQL (no version information provided) one but it better than FirebirdSQL documentation approach. Besides, many issues of MariaDB are compatible with MySQL, so if any information is missing in the documentation provided on the official MariaDB site, it is possible to find it from MySQL [82] documentation. Nevertheless, there is a potential risk of mismatching between MySQL and MariaDB supported features caused by growing furcation between the original (MySQL) and the fork (MariaDB). Thus, let us assume that FirebirdSQL is the least documented, MariaDB is the moderate documented and PostgreSQL is the most documented in this set of DBMSs.

The more popular DBMS has definitely bigger community. In November 2014 in the complete DBMS ranking FirebirdSQL Server on 26-th place and MariaDB on 27-th place [83], in relational only DBMS ranking FirebirdSQL Server on 15-th place and MariaDB on 16-th place [84]. PostgreSQL is in both ranking tables on 4-th place. However, as already suggested before MariaDB has very much in common with MySQL that is on the second place in both ranking. There is a lot of useful information about MariaDB on the

MySQL Performance Blog sponsored by Percona LLC [85] and Ronald Bradford's site Effective MySQL [86] who visited Tallinn on 27 August 2014 to meet with Estonian MySQL community. Ronald Bradford is Oracle ACE director and author of several books about effective implementation of MySQL. So taking into the account the big community of MySQL users and developers it is proposed considering MariaDB supportive community is bigger than PostgreSQL one. FirebirdSQL can be supposed as having the least community comparing with two others DBMS.

Thereby, PostgreSQL and MariaDB are estimated as equal in this section because first one is better documented and second can find better community support .The assessments of DBMS from documentation and community point of view are proposed below (Table 12).

Table 12. The comparison matrix of documentation and community.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/3	1/3
MariaDB	3	1	1
PostgreSQL	3	1	1

2.4.3. Mobility

In the beginning of Chapter 2.4 all three DBMSs were selected due to their mobility. Then under the definition of mobility meant platform independence. In this chapter, the DBMSs have been examined also from used disk sizes and deployment point of view.

In order to calculate occupied disk space for every DBMS has been created and populated already described OSDB database. All three DBMSs have been filled from the same 5 CSV files.

The OSDB database in FirebirdSQL is represented as one file with size 55.11 MB. Same database as sum of five tables with indices takes in PostgreSQL 76.2 MB and in MariaDB 88.2 MB. Consequently, FirebirdSQL has smallest footprint among the DBMSs. The essential difference between FirebirdSQL and the rest has been achieved due to the fact

that osdb database in this DBMS does not contain indices at all. However, even without indices PostgreSQL has total database size 66.1 MB and MariaDB has 74.1 MB database. However, database from DBMS cannot be just moved to the other system, it should be initially backed up. All three DBMSs can be backed up using appropriate backup utilities: pg_dump in PostgreSQL [46] and mysqldump for MariaDB [47], gbak for FirebirdSQL [8]. Then the dump files can be moved to other system and imported using psql (PostgreSQL) and mysql (MariaDB) DBMS interactive terminals, or already mentioned gbak utility for FirebirdSQL. Thus pg_dump has produced dump file that has been compressed to 32 MB using –Fc flags of the utility (Fig. 9).

```
osdb@linux-mlqn:/home/oleg/dumps> pg_dump -Fc osdb > pgosdb.dump
osdb@linux-mlqn:/home/oleg/dumps> ls -latr --block-size=M
total 32M
drwxr-xr-x 41 oleg users  1M Nov 20 11:17 ..
drwxrwxrwx  2 root root   1M Nov 20 12:16 .
-rw-r--r--  1 osdb users 32M Nov 20 12:16 pgosdb.dump
```

Figure 9. Creating PostgreSQL database dump

After creation FirebirdSQL and MariaDB dump files and compression them using gzip the least archived dump size has been achieved in case of MariaDB. The compressed dump of MariaDB myosdb.sql.gz has 23831 KB and the analog file in FirebirdSQL has 24989 KB (Fig. 10).

```
osdb@linux-mlqn:/home/oleg/dumps> /usr/bin/mysqldump -uroot -p -C  myosdb > /home/oleg/dumps/myosdb.sql
Enter password:
osdb@linux-mlqn:/home/oleg/dumps> ls -latr --block-size=K
total 108512K
-rw-r--r--  1 osdb users 31851K Nov 20 12:16 pgosdb.dump
-rw-r--r--  1 root root  24989K Nov 20 13:52 fb.fbk.gz
drwxr-xr-x 41 oleg users     4K Nov 20 14:57 ..
drwxrwxrwx  2 root root      4K Nov 20 15:29 .
-rw-r--r--  1 osdb users 51658K Nov 20 15:29 myosdb.sql
osdb@linux-mlqn:/home/oleg/dumps> gzip myosdb.sql
osdb@linux-mlqn:/home/oleg/dumps> ls -latr --block-size=K
total 80684K
-rw-r--r--  1 osdb users 31851K Nov 20 12:16 pgosdb.dump
-rw-r--r--  1 root root  24989K Nov 20 13:52 fb.fbk.gz
drwxr-xr-x 41 oleg users     4K Nov 20 14:57 ..
-rw-r--r--  1 osdb users 23831K Nov 20 15:29 myosdb.sql.gz
drwxrwxrwx  2 root root      4K Nov 20 15:30 .
osdb@linux-mlqn:/home/oleg/dumps>
```

Figure 10. The dump files' sizes

The obvious plus of mysqldump utility is also remarkable number of available options, there are 89 command line arguments, allowing the most flexible way of making back up. The minus is this is the least user-friendly utility amid competitors that requires higher professional skills. The essential disadvantage of FirebirdSQL is that ability to migrate from this DBMS to others, for example PostgreSQL or MariaDB, is limited. This gbak utility converts database to proprietary FirebirdSQL format that is not set of metadata and

SQL sentences like PostgreSQL and MariaDB dumps [48]. Few third-party tools like free FBExport [49] or commercial DBConvert [50] are able to assist in the migration from FirebirdSQL to other DBMS. The largest number of possibilities to switch from one DBMS to another is supplied for MariaDB. The mysqldump has as one of the command line arguments option "compatible" that can make dump more compatible with PostgreSQL database format [51]. Thus using this option and set of Perl scripts it is possible to convert existing MariaDB database dump to format that is readable for PostgreSQL. There are also numerous CLI and GUI tools, one of them is MySQL Workbench [52] that can execute migration from MariaDB to other DBMS formats. The same tool can be used to migrate from PostgreSQL to MariaDB (MySQL) database. The total list of available tools to migrate from PostgreSQL is significantly shorter. Nevertheless, because PostgreSQL dump as already mentioned consist of SQL commands and metadata it can be read using standard text processors and utilities like cat, less, tail or head in Linux with latter structural analyze and manual import.

Taking into the account that MariaDB is most feature-rich with the least compressed dump size it is proposed to concern this DBMS as most mobile between three participants. PostgreSQL has less options then MariaDB to dump and biggest file size, but with open file format. Therefore, the second place in the mobility examination belongs to this DBMS. FirebirdSQL with its non-readable dump file format and limited number, comparing to two others, of migration possibilities is on the last place (Table 13).

Table 13. Mobility comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/5	1/3
MariaDB	5	1	3
PostgreSQL	3	1/3	1

2.4.4. Reliability

As was already mentioned above under reliability of DBMS is usually meant an assessment composed of security, backup and restoring issues. DBMS security concerns authentication, authorization, data confidentiality and integrity.

However, in the beginning the relevance to ACID principled of the DBMSs need to be concerned. All three DBMS have sufficient MVCC support [ref 8]. Nevertheless, MariaDB as representative of MySQL family DBMS ha the only transactional data storage engine InnoDB. The predecessor of Firebird InterBase was first DMBS where the MVCC was realized [12]. PostgreSQL is fully transactional DBMS with its own implementation of MVCC. All three DBMSs support two-phase commit protocol [18], including MariaDB InnoDB, and regarding to transaction durability can be assessed as equal.

Authentication mechanism of PostgreSQL is most flexible and complicated one among these three DBMSs. The three components of authentication are host, name and method of authentication. The combination of these components defines how different users from different hosts can connect to server and what identification methods they have to use [9]. The users can be allowed to connect using passwords and encrypted passwords, Kerberos identification system, identification card. It is possible as well to explicitly connect without password identification or reject any connection from certain host. PostgreSQL supports user groups for facilitation user rights management. PostgreSQL has integrated LDAP support that can be implemented authentication purposes. Authentication mechanism of MariaDB based also on host and user names without definition of access method. The only way to identify the user is to provide the user's password. There is no group support mechanism in MariaDB. LDAP support provided by PAM (Pluggable Authentication Modules) that included in installation package since version 5.2.10 [11]. This is the meaningful difference with MySQL that does not have integrated support of LDAP. The simplest authentication way is applied in FirebirdSQL server where users identified by user name and password. The password has maximum length eight bytes that is definitely unsufficient. There is one more significant security fault - in Posix systems embedded clients can at least see security database in case of Classic Server configuration [1].

Authorization in both PostgreSQL and MariaDB is concentrated to maintain two substantial aspects of this process. The first is ability to guaranty access to specific object of a database like table, procedure, trigger and even column using standard SQL GRANT/REVOKE statements. The second one covers database administrative and maintenance tasks like creating databases and analyzing tables that is permitted by default to DBA. In case of FirebirdSQL server newly created user is able to create his/her own database object and populate it with tables, procedures and so on. In this work this way was successfully tested, in other words new user created in FlameRobin FirebirdSQL administration tool, then logged in using isql-fb utility and new database and table was created. In two others DBMSs the privileges to create database are not implicitly permitted, that is benefit from security point of view. FirebirdSQL has standard GRANT/REVOKE procedures that can be applied to modify the rights of user. In addition, FirebirdSQL supports UNIX/Linux groups and accounts.

PostgreSQL has built in support of SSL cryptographic protocol that allows establishing secure connections, it is possible to implement SSH and Stunnel protocols for channel encryption. MariaDB has also support of SSL that is disabled by default, but can be activated by change “have_ssl” variable [16]. There is also possibility to use SSH and Stunnel protocols to encrypt channel for MariaDB. The built in support of any encryption protocol is missing in FirebirdSQL [17]. Developers propose to use third-party ZeebeDee software that realizes its own protocol to create encrypted channel or standard SSH-compatible tools [17].

The backup and restoration utility of FirebirdSQL gback is able to back up without interruption of the database work [8]. The utility has impressive number of available options for both backup and restoration. There is another administrative utility supplied with DBMS gfix that intended to use for database fixing, sweeping (garbage collection) and other tasks. Percona XtraBackup tool is free tool for MariaDB, MySQL and Percona Server hot backup [20]. PostgreSQL has supplied with DBMS utilities that can be used for hot backup (pg_dump) and restoration (pg_restore) [9]. Accordingly, all three DBMSs have equal capabilities for backup and restoration. However, only two of them – PostgreSQL (WAL system) and MariaDB (with InnoDB storage engine) have automatic recovery. Automatic consistency check and recovery for FirebirdSQL is provided by third-party software [8].

Summarizing the comparison of the DBMSs' reliability it is necessary to mention that only in transaction durability research all three systems were represented with certain degree of equality (for deeper and more comprehensive analysis all three MVCC mechanisms should be tested). In others components of assessment the leading position belongs to PostgreSQL, MariaDB is on the second and FirebirdSQL is on the last third position.

Consequently, the DBMS reliability comparison matrix can be filled as following (Table 14).

Table 14. Reliability comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/3	1/5
MariaDB	3	1	1/3
PostgreSQL	5	3	1

2.4.5. Functionality

All three examined DBMSs provide high level of functionality required for SME, yet there are differences. The DBMSs have triggers can be fired on INSERT, UPDATE and DELETE operation events (defined in SQL standard) with BEFORE and AFTER timing preconditions. In addition, PostgreSQL supports more operations like TRUNCATE commands but only FOR EACH statement [21]. If there are constraints like foreign key are defined by CREATE TABLE operation, PostgreSQL produces constraint triggers. The additional type of supported preconditions supported by PostgreSQL is INSTEAD OFF. In PostgreSQL triggers and stored procedures can be written in PL/pgSQL PL that is similar to Oracle PL/SQL. A trigger in PostgreSQL fires trigger function that can be written on PL/pgSQL or C or other supported PL. There is a core of PL included in standard PostgreSQL distribution. It consist of PL/pgSQL, Pl/Perl, PL/Tcl, and PL/Perl [22]. Besides, seven PL language are developed independently for PostgreSQL: PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, and PL/sh [23]. Triggers in PostgreSQL can be applied to different table and even view columns. On the contrary, MariaDB triggers have substantially more limited functionality originated from MySQL. For

example, triggers cannot be fired by foreign key actions [24] that means if there is a trigger on child table and the table has been modified due to CASCADE rules (DELETE OR UPDATE) the trigger misses this event and child table remains unmodified. Moreover, only SQL statements can fire the trigger. In other words, if API implementation does not imply direct transmitting of SQL statements the trigger misses table modification [25]. This lack of functionality can lead to inconsistence of data and can also be considered in previous chapter 2.5.4. as reliability gap. The next limitation is the limited number of allowed triggers in table for each type of timing/events, thus if a table already has a trigger fired with combination AFTER UPDATE a new trigger with same combination regarding to the same table cannot be added. MariaDB triggers cannot be applied to specific columns. Meantime triggers of PostgreSQL activate UDF, MariaDB triggers react only by mix of SQL commands and built in functions like SYSDATE. FirebirdSQL Server does not have MariaDB trigger limitations. For instance, there is no limitation of trigger type number in one table [26]. Moreover, triggers in FirebirdSQL can be created to react on database events. The events are CONNECT, DISCONNECT, TRANSACTION START, TRANSACTION COMMIT, TRANSACTION ROLBACK [27]. In both PostgreSQL and FirebirdSQL have multiple event statement composed by OR statements that can fire triggers. MariaDB trigger can be fired by one only event. Stored procedures can be written in FirebirdSQL with PSQL - so-called procedural SQL [8], C and Pascal can be used to create UDFs [28]. Stored procedures of MariaDB as MySQL fork are based syntax is defined in SQL: 2003 [29]. MariaDB supports writing of UDFs in C and C++, but it is possible to implement Perl as well [30]. Thus, the most functional DBMS in the scope of triggers, stored procedures and UDF is PostgreSQL. MariaDB could be considered as less functional in this context and FirebirdSQL is between them due to its limited support of PLs.

All three DBMSs support different client APIs : C, C++, ODBC, JDBC, Python, Perl, Ruby, PHP, Shell and .NET. Therefore, from supported API point of view it is proposed to concern all three as equal to each other.

In practice, during working on osdb-fb project (OSDB version for FirebirdSQL) the serious lack of functionality has been faced. Meanwhile PostgreSQL and MariaDB have built in capabilities to import large volumes of data, in FirebirdSQL, large text or CSV files are proposed to import from external file [8].

```
CREATE TABLE tenpct external file '/db/asap.tenpct'
(
   col_key CHAR(8),
   C1 CHAR(1),
   col_int CHAR(8),
   C2 CHAR(1),
   col_signed CHAR(10),
   C3 CHAR(1),
   col_float CHAR(12),
   C4 CHAR(1),
   col_double CHAR(9),
   C5 CHAR(1),
   col_decim CHAR(9),
   C6 CHAR(1),
   col_date CHAR(9),
   C7 CHAR(1),
   col_code CHAR(10),
   C8 CHAR(1),
   col_name CHAR(20),
   C9 CHAR(1),
   col_address VARCHAR(80),
   NL CHAR(1)
);
COMMIT;
```

Figure 11. Creating table in FirebirdSQL using external file.

This import can be executed using only fixed length of columns (Fig. 11), so if a file contains data with variable length the data can be shifted to neighbor column (Fig. 12).

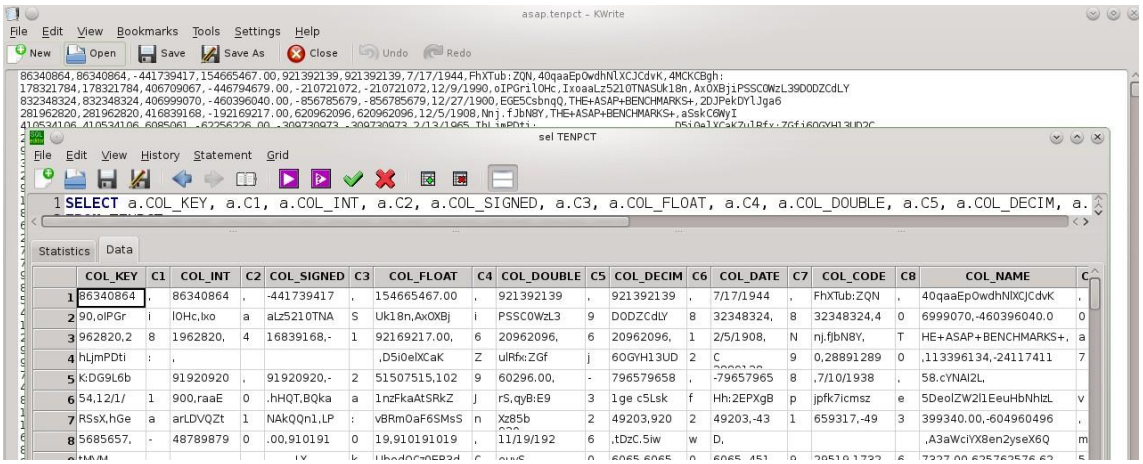

Figure 12. Initial CSV file and imported table.

This was the reason to use C++ boost library to add custom import method to the project. The method has been described in Appendix 2 “Architecture of experimental platform”.

In addition, during benchmarking of FirebirdSQL “bulk_append” test with SQL query "INSERT INTO updates SELECT * FROM updates" was not finished even after 12 hours of computation. The AS3AP recommends that every test have to be finished within 12

hours [6]. That is the reason why this “bulk_append” test considered as a failed for FirebirdSQL and has been further excluded from overall test sequence.

Moreover, FirebirdSQL is not able to create hashed table indexes. For this reason, the next benchmark tests cannot be performed for this DBMS:

1. “create_idx_hundred_code_h”,
2. “create_idx_tenpct_code_h“,
3. “create_idx_tenpct_code_h”,
4. “create_idx_tenpct_name_h”,
5. “create_idx_uniques_code_h”,
6. “create_idx_updates_code_h”

All described above shortcomings discovered due to benchmarking creation should be kept in mind during DBMSs’ comparing. As result proposed table is represented below (Table 15).

Table 15. Functionality comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/3	1/5
MariaDB	3	1	1/3
PostgreSQL	5	3	1

2.4.6. Scalability

Meanwhile MariaDB 10.1 demonstrates better performance than MySQL [31] that is able to support up to 48 processor cores since version 5.6 [32], PostgreSQL supports up to 64 cores from version 9.2 [33]. Both MariaDB and PostgreSQL have built-in replication mechanisms and number of open source solutions that allow efficient distributing and balancing the load. Introduced in version 9.0 of PostgreSQL Streaming Replication (SR) named features supplies WAL log records from master server to slave (or standby) servers [34]. This principle is very similar to replication mechanism implemented in MySQL (and what MariaDB uses) since version 5.1 [35]. Before replication in PostgreSQL was

realized on triggers (Slony) and statement-based middleware (Pgpool-II) and was significantly slower than MySQL (and MariaDB) had [36]. Now PostgreSQL and MariaDB have equal horizontal scalability with warm and hot standby servers, with different topology options including multi-master support - with MariaDB 10.0 [37] and Postgres-XC [38]. FirebirdSQL does not have any built-in replication support, but there are third-party tools available for this purpose [39]. The most advanced of them SymmetricDS is open source platform independent tool that supports multi-master replication for number of DBMSs including FirebirdSQL, MariaDB and PostgreSQL [40]. This tool is trigger-based and can be used for asynchronous replication only. Firebird Classic and SuperClassic versions support SMP [41, 42], so the DMBS can be scaled up although exact data how many processor cores can be supported is not available.

All three DBMSs have no restriction concerning maximum database size. MariaDB has maximum table size 64 TB in case of InnoDB storage engine [43]; meanwhile PostgreSQL [44] and FirebirdSQL [45] have maximum table size 32 TB. Nevertheless, PostgreSQL have the largest maximum row size 1.6 TB [44] comparing to the rest – both MariaDB (InnoDB) and FirebirdSQL have 64 KB maximum row size [43, 45].

In a scope of scalability issue FirebirdSQL is slightly behind the rest of examined DBMSs. PostgreSQL and MariaDB can be estimated as equal. The results of assessment are in Table 16.

Table 16. Scalability comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/3	1/3
MariaDB	3	1	1
PostgreSQL	3	1	1

2.4.7. Performance

The results of OSDB benchmarking presented in Table 15. demonstrate the time period spent for every test. The architecture of the benchmarking have been described previously in Chapter 2.2. The expressions of the test queries reside in Appendix 1.

As already mentioned in Chapter 2.5.5, that some of the tests are not available in case of FirebirdSQL DBMS due to its functional dearth.

It was already mentioned in Chapter 2.5.5 that FirebirdSQL has no standard functions to upload information from text file into the database table and for this purposes this feature has been implemented on the client side. The implementation has led to drastically enlarged "Dataset load" test time with its more than 448 seconds spent to populate the tables. As was mentioned in 2.5.5 that some other tests marked as failed and N/A have been failed and unavailable for FirebirdSQL.

Table 17. The performance testing results (seconds)

Test name	**PostgreSQL 9.2.7**	**MariaDB 5.5.33**	**FirebirdSQL 2.5.2**
Database creation	0.842518	0.00269389	0.185572
Table creation	0.0512509	0.888244	2.71767
Dataset load	2.48202	20.3042	448.251
agg_create_view	0.00862002	0.143817	0.390566
agg_func	0.0666161	0.113545	0.0581758
agg_info_retrieval	0.034838	0.0867929	0.049927
agg_scal	0.034066	0.048306	0.0332489
agg_subtotal_report	0.214588	18.8833	0.0332561
agg_total_report	0.197711	18.8765	0.016845
bulk_append	0.478797	3.2737	failed
bulk_delete	0.04476	0.245332	0.0994558
bulk_modify	0.220317	0.242565	9.75764
bulk_save	0.0539119	0.112962	0.0169928
count_tuples	0.052794	0.110906	0.0165181
create_idx_hundred_code_h	0.177024	1.1546	N/A
create_idx_tenpct_code_h	0.183337	1.30896	N/A

Test name	**PostgreSQL 9.2.7**	**MariaDB 5.5.33**	**FirebirdSQL 2.5.2**
create_idx_tenpct_name_h	0.533135	1.91246	N/A
create_idx_uniques_code_h	0.183253	0.749002	N/A
create_idx_updates_code_h	0.374942	1.86958	N/A
join_2	0.05656	0.131412	0.23788
join_2_cl	0.0199289	0.116559	0.182784
join_3_cl	0.0207541	0.119782	0.266508
join_3_ncl	0.000867128	0.000396013	0.291473
agg_func	0.0203559	0.111908	9.41249
join_4_ncl	0.000828981	0.00039506	0.374785
proj_100	4.09898	0.140239	0.254043
proj_10_pct	0.11451	0.116066	0.138842
sel_100_cl	0.0656519	0.153201	0.0153391
sel_100_ncl	0.043731	0.151931	0.0164399
sel_10pct_ncl	0.000550032	0.000246048	0.0166061
sel_1_cl	0.0375061	0.149212	0.0165019
sel_variable_select_high	0.0839369	0.109101	0.01653
sel_variable_select_low	0.0221109	0.075038	0.0166512
table_scan	0.0216441	0.104612	0.0165169
upd_append_duplicate	0.0233419	0.000787973	0.132842
upd_app_t_end	0.00815701	7.82013e-05	0.116557
upd_app_t_mid	0.00829101	0.00012207	0.199832
upd_del_t_end	0.037215	0.22138	0.116531
upd_del_t_mid	0.0543849	0.228939	0.133225
upd_integrity_test	0.0250139	0.1102	0.0998979
upd_mod_t_cod	0.0381589	0.222935	0.10823
upd_mod_t_end	0.0449889	0.221459	0.0999041
upd_mod_t_int	0.038198	0.222496	0.108262
upd_mod_t_mid	0.0378802	0.222431	0.0998709
upd_remove_duplicate	0.0403731	0.222935	0.099906

The Appendix 4 contains the results benchmarking tests and results deviation table. In the table, the best result of a test is presented with green, the worst with red and moderate with default black color.

According to these results, the best performance has been undoubtedly achieved in case of PostgreSQL benchmarking. This DBMS has demonstrated the best results in 27 and the worst only in 2 benchmarks. MariaDB DBMS has not failed any test from the OSDB benchmark suit, but it has shown only 8 (10 in case of FirebirdSQL) the best results and 22 the worst results (21 FirebirdSQL). Nevertheless, tremendous difference in dataset load test, caused by absence in Firebird native method to load big data array directly to table, gives the basement to assume that MariaDB has better performance than FirebirdSQL Server does. The results of performance comparison presented below (Table 20).

Table 18. Performance comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/3	1/5
MariaDB	3	1	1/3
PostgreSQL	5	3	1

All program code (including original osdb-0.90) created in the framework of this thesis reside on the attached disk.

2.4.8. Power efficiency

In order to evaluate power efficiency of DBMSs it is assumed that every DBMS should execute same benchmarks during one hour. For this purposes some benchmark tests have been excluded from MariaDB and PostgreSQL because FirebirdSQL is not able to run these tests. Thereby, the number (37 benchmark tests) and tests themselves are same for all three DBMSs. Code of all three projects has been also modified, now all three DBMSs don't create, populate and drop databases and code includes only the tests compatible with FirebirdSQL Server's functionality.

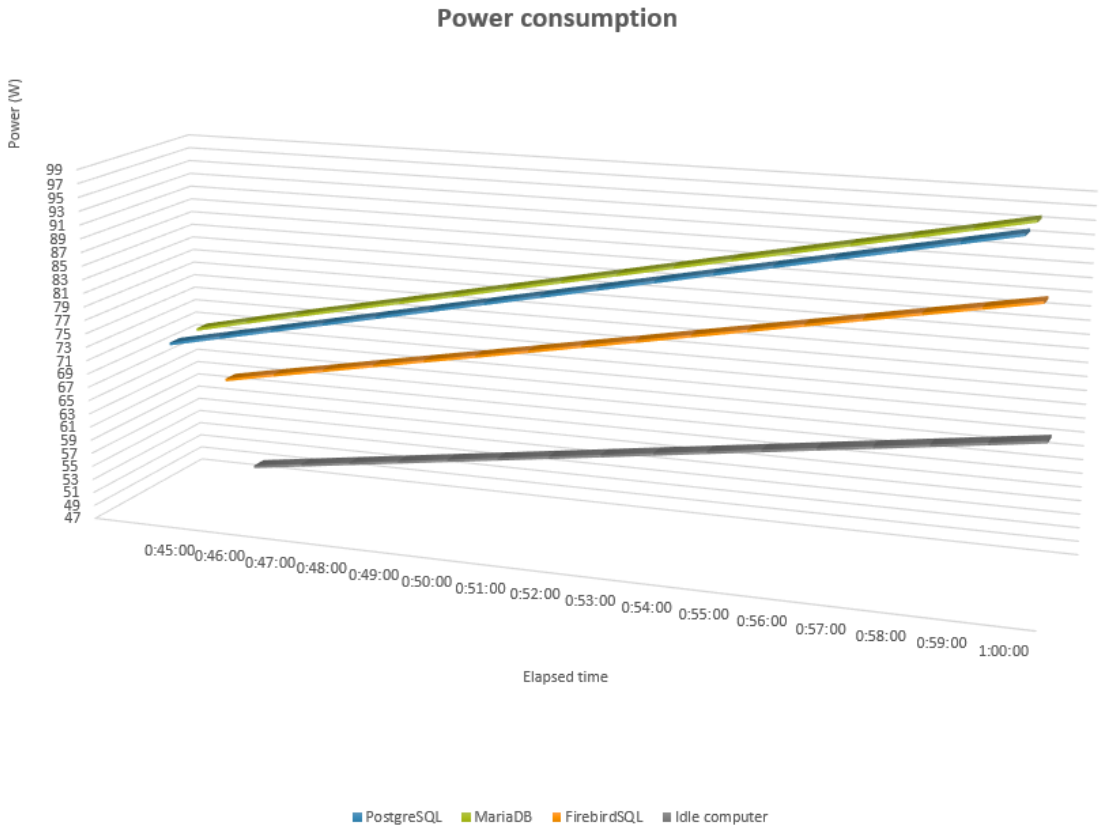

Figure 13. Comparison of power consumption.

In PostgreSQL hour operation test measured by KYORITSU KEW 6310 analyzer power consumption is 98.469 W (all data of power measurement resides in Appendix 5). The number of executed transactions is 93573. According to proposed equation (2) the power efficiency of PostgreSQL DBMS on the current experimental system is:

$$\mu = \frac{N\ tr}{tPc} = \frac{93573}{1*98.469} = 950.279\ (TpW) \qquad (3)$$

The results of one-hour-benchmarking of MariaDB demonstrate the number of executed transactions is 200466. During the benchmarking, the experimental system consumed 98.6856 W. Thereby, the power efficiency of MariaDB can be evaluated as:

$$\mu = \frac{N\ tr}{tPc} = \frac{200466}{1*98.6856} = 2031.360\ (TpW) \qquad (4)$$

The one-hour-benchmarking of FirebirdSQL has discovered the least number of executed transactions between all tests - 33041. The system consumed 86.1914 W. Thus, the evaluated power efficiency of FirebirdSQL:

$$\mu = \frac{N\ tr}{tPc} = \frac{33041}{1*86.1914} = 383.346\ (TpW) \qquad (5)$$

The power consumption of idle system has been measured and compared with three executed benchmarks (Figure 13).

The comparison table (Table 21) depicts superiority of MariaDB over the competitors in area of power efficiency.

Table 19. Power efficiency comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	1	1/7	1/5
MariaDB	7	1	3
PostgreSQL	5	1/3	1

3. Synthesis of the methodology

3.1. Calculation

In this part of work, it is proposed to use third method of normalization described in Chapter 1.5 to calculate the priority values of the matrixes.

The sums of Table 13 columns are 9.000 (first column), 1.533 and 4.333. After dividing the values in cells by appropriate sum, the normalized matrix has been calculated (Table 20).

Table 20. The normalized matrix of management tools.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.111	0.130	0.077
MariaDB	0.556	0.652	0.692
PostgreSQL	0.333	0.217	0.231

The average value of a row represents the priority value. Thus, the FirebirdSQL Server's priority value is 0.106, the priority of MariaDB is 0.633 and PostgreSQL has 0.260 as priority value.

The column sums of Table 14 are 7.000, 2.333 and 2.333. The normalized table has been calculated in the same way (Table 21).

Table 21. The normalized matrix of documentation and community.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.143	0.143	0.143
MariaDB	0.429	0.429	0.429
PostgreSQL	0.429	0.429	0.429

The priorities are 0.143 (FirebirdSQL), 0.429 both for MariaDB and PostgreSQL.

The column sums of Table 15 are 9.000, 1.533 and 4.333.

Table 22. The normalized mobility matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.111	0.130	0.077
MariaDB	0.556	0.652	0.692
PostgreSQL	0.333	0.217	0.231

Consequently, the priorities are 0.106 (FirebirdSQL), 0.633 (MariaDB) and 0.260 (PostgreSQL).

The column sums of Table 16 are 9.000, 4.333 and 1.533.

Table 23. The normalized reliability comparison matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.111	0.077	0.130
MariaDB	0.333	0.231	0.217
PostgreSQL	0.556	0.692	0.652

The calculated priorities are 0.106 (FirebirdSQL), 0.260 (MariaDB) and 0.633 (PostgreSQL).

The results of functionality comparison are identical with the results reliability comparison, it means that using the same normalization method the normalized matrix is equal with reliability one and as result the priority vector are the same: 0.106 (FirebirdSQL), 0.260 (MariaDB), 0.633 (PostgreSQL).

Using the same method for three the rest evaluation criteria have been calculated the next priorities for them.

Table 24. The normalized scalability matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.143	0.143	0.143
MariaDB	0.429	0.429	0.429
PostgreSQL	0.429	0.429	0.429

The scalability priorities are 0.143 (FirebirdSQL), 0.429 (both MariaDB and PostgreSQL).

Table 25. The normalized performance matrix.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.111	0.130	0.077
MariaDB	0.556	0.652	0.692
PostgreSQL	0.333	0.217	0.231

The performance priorities are 0.106 (FirebirdSQL), 0.633 (MariaDB) and 0.260 (PostgreSQL).

Table 26. The normalized matrix of power efficiency.

	FirebirdSQL	MariaDB	PostgreSQL
FirebirdSQL	0.077	0.097	0.048
MariaDB	0.538	0.677	0.714
PostgreSQL	0.385	0.226	0.238

The priorities of power efficiency are 0.074 (FirebirdSQL), 0.643 (MariaDB) and 0.283 (PostgreSQL).

As the result the priorities can be collected to the matrix depicted in Table 29 (used described in Table 2).

Table 27. The priorities of the criteria.

	Tools	**Docs**	**Mob**	**Rel**	**Fun**	**Scl**	**Perf**	**Power**
FirebirdSQL	0.106	0.143	0.106	0.106	0.106	0.143	0.106	0.074
MariaDB	0.633	0.429	0.633	0.260	0.260	0.429	0.260	0.643
PostgreSQL	0.260	0.429	0.260	0.633	0.633	0.429	0.633	0.283

This matrix has been multiplied with the vector of priorities (Table 7) according to mathematical rules of multiplying matrices. The result of this operation is the vector of DBMS ranking (Table 28).

Table 28. The vector of DBMS ranking.

FirebirdSQL	0.107
MariaDB	0.395
PostgreSQL	0.497

The largest value in this vector belonging to PostgreSQL determines the DBMS that is most suitable for selection criteria important for Estonian SMEs. The least value has been achieved in case of FirebirdSQL Server defines this DBMS as the least appropriate from the three examined DBMSs for implementation in circumstances defined in this work as important for SME. The implementation capabilities of MariaDB are evaluated as moderate.

3.2. Analysis Results

The results of the experimental evaluation have revealed the best DBMS in the scope of proposed generalized set of criteria. In the meantime, the criteria could vary from one business area to another. Thus, the requirements to DBMS of webhosting enterprise are definitely different from transport company's ones. Probably for a webhosting company the power efficiency issue can be more is the one of the most important. In this case, the comparison matrix of the selection criteria (Table 4) can be adopted according to the company requirements and as result the most power efficient DBMS can be selected. In other words, the proposed methodology allows to be ad hoc accommodated.

The investigation of CPU temperature during the intensive load has revealed the fact concerning utilization of the CPU cores in the benchmark test. There is a program named Psensor can be implemented for CPU temperature monitoring in Linux systems. Measuring CPU temperature with Psensor it has been discovered that temperature graphs in PostgreSQL and FirebirdSQL benchmark tests look similar meanwhile MariaDB benchmarking has produced the temperature graph that significantly varies from others (Appendix 6). The Psensor has displayed the temperature of both CPU cores by red (Core 0) and blue (Core 1) colors. In case of PostgreSQL and FirebirdSQL the temperature of CPU core has been declining every time after reaching the maximum temperature value while the temperature of the second core has been arising. In the next measurement period, the temperature that had been growing before fell while another one was growing. Meantime the temperature graph produced by MariaDB demonstrates that temperature of the one core was in its maximum point during several measurement period. The architecture features of the DBMSs could cause such behavior. PostgreSQL and FirebirdSQL SuperClassic have architecture supporting SMP in full range. In spite of the fact that FirebirdSQL SuperClassic uses threads, not processes like PostgreSQL does, for client connections this DBMS uses more than one CPU core to handle results. It should be taken into account that there is one connection has been used in benchmarking. MariaDB like others MySQL representatives (MySQL, Percona Server) uses one thread per client connection. After thorough research it has been discovered that MariaDB does not change CPU core every time between benchmark cycles and prefers to run on the one core of CPU. Probably it can be explained as using custom thread pool in its architecture that does not return this resource completely to OS, but uses it in the new connection. There is at least one benefit when thread pool is implemented – the overhead related

thread creation. Nevertheless, the difference between temperatures of two cores has achieved at single moments up to 7°C and has lasted for minutes. Besides, the maximum temperature (Appendix 6) achieved in case of MariaDB 80°C exceeds maximum temperature achieved in PostgreSQL benchmarks by 5°C and FirebirdSQL by 12°C. The average temperature in MariaDB test is also higher than in PostgreSQL and FirebirdSQL benchmarks and reaches maximally allowed by the manufacturer 72 °C. The power consumption of the used Intel® Core™ 2 Duo CPU E6550 at this temperature is 65 W [14]. The average temperature achieved in PostgreSQL tests is about 70 °C (about 61 W) and 64 °C (46 W) in FirebirdSQL tests [14]. As can be noticed here the homogenous distribution of the load between CPU cores facilitates following the thermal profile recommended by the manufacturer and is more important than temporal benefits from minimizing the OS resource allocation.

3.3. Possible improvements in the methodology

3.3.1. Power efficiency evaluation

It is possible to improve the equitation of power efficiency evaluation (2). Power consumption of examined idle system can be evaluated and deducted from the amount of consumed power. The improved equitation is presented below (6).

$$\mu = \frac{N\ tr}{t(Pc-Pi)} \qquad (6)$$

In this equitation N_{tr} is the number of executed transactions, t is time in hours, P_c is the power consumed by a system under running benchmarks and P_i is the power of the idle system. Thereby the calculation of power efficiency based only on the amount of power that has been consumed for transactional activity of a DBMS. Hence, power efficiency can be evaluated accordingly for MariaDB (7), PostgreSQL (8) and FirebirdSQL Server (9).

$$\mu = \frac{N\ tr}{t(Pc-Pi)} = \frac{200466}{1\times(98.6856-64.6793)} = 5894.984\ (TpW) \qquad (7)$$

$$\mu = \frac{N\ tr}{t(Pc-Pi)} = \frac{93573}{1\times(98.469-64.6793)} = 2769.276\ (TpW) \qquad (8)$$

$$\mu = \frac{N\ tr}{t(Pc-Pi)} = \frac{33041}{1\times(86.1914-64.6793)} = 1535.926\ (TpW) \qquad (9)$$

As can be noted here, removing idle consumption allows the equitation to describe the power efficiency more exactly. FirebirdSQL was slower in the tests, but it also consumed less power.

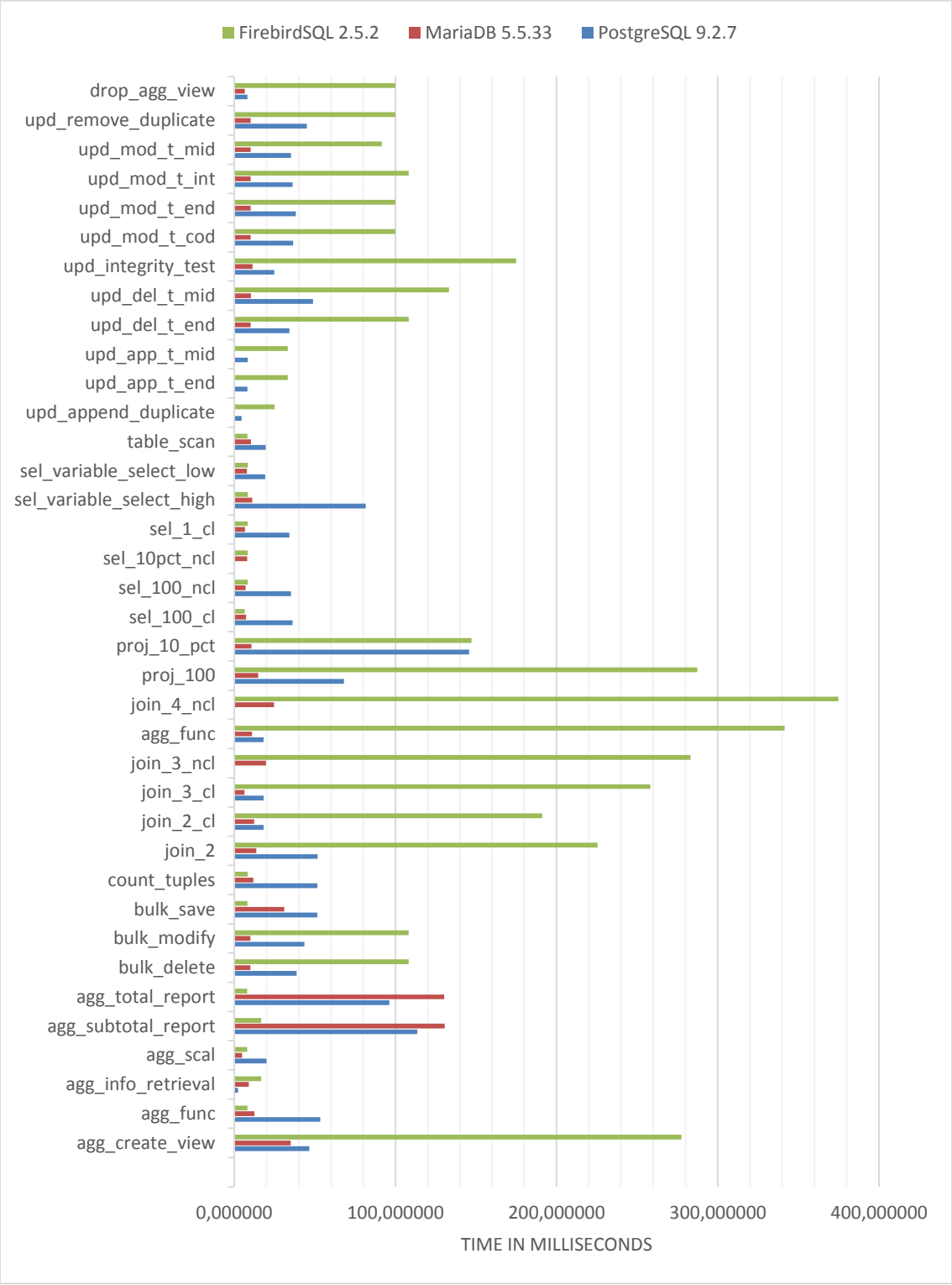

Figure 14. Performance of limited benchmark set.

The results depicted on Figure 14 have been got during described in Chapter 2.4.8. power efficiency test. This test has been adopted to be able to evaluate power efficiency of all three selected DBMSs. In order to do this evaluation possible to demonstrate the methodology, the initial list of the OSDB queries has been cut by the reason of limited functionality of FirebirdSQL. Despite of this FirebirdSQL has lagged behind the competitors significantly. In addition, it should be kept in mind that AS^3AP recommendations consider benchmarking process as comprehensive [6] so all predefined benchmarks ought to be included. In this case, PostgreSQL that is able to load dataset in 2.48 seconds can be considered as more power efficient DBMS than MariaDB that loads same dataset in 20.32 seconds. Hereby it is recommended to use maximum sized set of benchmarks allowing more complete testing of DBMS.

3.3.2. Common methodology

There is an opinion that AHP based methodology has some kind of uncertainty originated from expressing subjective assessments to the exact numbers and there is a better way to evaluate DBMS selection criteria [10]. This method can be considered as supplement of AHP methodology named as FAHP (Fuzzy Analytical Hierarchy Process). There are researchers elaborated the FAHP based DBMS selection model for Turkish National Identity Card Management Project [10]. Comparing to its predecessor AHP that uses exact ratios, the assessments in FAHP have to be done using fuzzy comparison ratios. The judgments of experts on one of the selection criteria can be collected for further operations, thus the judgments form triangular fuzzy number (TFN) [13]. The TFN can be expressed by equation (10).

$$T_{xy} = (L_{xy}, M_{xy}, H_{xy}), \text{ where } L_{xy}, M_{xy}, H_{xy} \in (\tfrac{1}{9}, 9) \qquad (10)$$

The L_{xy} and H_{xy} are respectively the lowest and the highest possible values [13]. The value of M_{xy} can be calculated as (11):

$$M_{xy} = \sqrt[n]{J_{xyn}}, \text{where } J_{xyn} = J_{xya} \times J_{xyb} \times J_{xyc} \ldots \times J_{xyn} \qquad (11)$$

There is a number n of respondent judgments J_{xy} and mathematically M_{xy} represents value where membership function of fuzzy set is equal to one [10]. Instead of the judgment

collection from different experts the comparison set of sub criteria can be applied to calculate M_{xy}. For instance, the issue of scalability can be compared as fuzzy set of its sub criteria, thus sub criterion maximum database size can be estimated by 5 meanwhile replication capability only by 3. In this case, the TFN is (3, 3.87, 5). The process of defuzzification converts magnitudes of fuzzy set to values. There are many methods to perform this process. The equation (12) based on alpha cut approach [13].

$$\mu_{\alpha,\beta}(\dot{F}_{xy}) = [\beta \times f_{\alpha}(L_{xy}) + (1 - \beta) \times f_{\alpha}(H_{xy})],\ where\ 0 \leq \alpha, \beta \leq 1 \qquad (12)$$

In this equation $\dot{F}_{xy}$ represents fuzzy pairwise comparison matrix. The sub equation (13) describes the lowest boundary value of alpha cut and the sub equation (14) the highest one [13].

$$f_{\alpha}(L_{xy}) = (M_{xy} - L_{xy}) \times \alpha + L_{xy} \qquad (13)$$

$$f_{\alpha}(H_{xy}) = H_{xy} - (H_{xy} - M_{xy}) \times \alpha \qquad (14)$$

The coefficients α and β used in the last three equations (12, 13, 14) represent preferences and risk tolerance [13]. The values for this coefficient should be selected basing on the principle of estimation the uncertainty degree in decision making, if this degree is high the values of the coefficients will be smaller than in case of lower degree of uncertainty [13]. The value 0.5 is neutral [13]. It means respondents estimate their judgments neither pessimistic nor optimistic [13]. Thereby, in case of already cited TFN equals to (3, 3.87, 5) with α and β both equal to 0.5 the calculated value of corresponding entry of comparison matrix is 3.935 (15).

$$\mu_{\alpha,\beta}(\dot{F}_{xy}) = [0.5 \times ((3.87 - 3) \times 0.5 + 3) + (1 - 0.5) \times (5 - (5 - 3.87) \times 0.5)] = \\ = 3.935 \qquad (15)$$

Then the comparison matrix can be normalized using regular methods described in Chapter 1.5. There is reasonable remark that greater number of the judgments leads to calculation of M_{xy} producing the value representing assessment that is more comprehensive. As consequence, the overall evaluation result will be more accurate than AHP is able to represent.

The second possible improvement can be done by thorough research of accordance of OSDB proposed benchmarks (Appendix 1) with AS^{3}AP recommendations. The deep analysis of used benchmark queries can spot some possible improvements. The authors of OSDB project declared that AS^{3}AP needs more SQL specifications rather than native language descriptions [7]. There is an ambiguity in AS^{3}AP documentation [6] concerning bulk queries block – in bulk_save and bulk_append queries is proposed to insert and select data from the relation named "saveupdates" that never mentioned before. These queries need to be replaced with others providing better compliance with AS^{3}AP recommendations.

4. Conclusion

In this thesis, the methodology for multi criteria selection of database management system has been elaborated and described. The selection criteria have been proposed considering the needs of Estonian SMEs. There are suggestions to use the next DBMS features as important selection criteria: the available management tools, documentation and community support, mobility, reliability, functionality, scalability, performance and power efficiency. The AHP has been implemented as the mathematical basis of the elaborated methodology. Every selection criterion has got its priority weight that has been used latter in the calculation of the final result. The methodology has successfully demonstrated its viability and one of three DBMSs, which have been analyzed in evaluation process, has declared as most suitable for Estonian SMEs. It should be mentioned here that priority weights of selection criteria have been calculated according to generalized needs of SMEs and ought to be recalculated for special cases. There is another multi criteria selection method reviewed in the analysis part of the thesis. This method FAHP is based on already known AHP and uses fuzzy sets for criteria evaluation. The FAHP is able to calculate an entry of comparison matrix using multiple judgments on the same subject getting more accurate input data than AHP does.

The OSDB open source project has been implemented in this thesis for DBMS performance evaluation purposes. This project is based on ANSI SQL Standard Scalable and Portable Benchmark for Relational Database Systems (AS^3AP) therefore can be used only for relational DBMS (RDBMS) performance assessment. The OSDB is written on C++ programming language and has been accommodated in the scope of the thesis for benchmarking of two DBMSs (PostgreSQL and FirebirdSQL Server). This adaptation revealed as well some issues about functional capabilities of these DBMS that have been taken into account for functionality assessment. Besides, the benchmarks based on the OSDB code have been used in the power efficiency evaluation part, where the benchmarks have been running during the one hour performing transactions. The codes have been slightly modified for the power efficiency tests.

In the beginning of the thesis it is suggested for evaluation of power efficiency to use the relation of executed transaction number and consumed power in watts. The proposed measurement unit is named as Transactions-per-Watt (TpW) in parallel with famous TPC (Transactions-Per-Cent) benchmark tests for proprietary server systems. In the analysis

part, the proposed method for power efficiency evaluation has been improved by subtracting the consumption of the idle system from the value consumed by the system with running benchmarks. The power quality analyzer KYORITSU KEW 6310 has been used in this thesis for power consumption measurements.

The elaborated methodology can be considered as consisting of three parts – an overall estimation system for multi criteria selection, the DBMS performance benchmarking and the power efficiency assessment. Besides, every part can be used separately. The AHP (or FAHP) method of multi criteria selection can be implemented for estimation both relational and non-relational DBMSs. The method of DBMS power efficiency evaluation can be implemented for all types of DBMS and others high load applications. However, the AS^3AP based DBMS performance benchmarking needs to be replaced to appropriate for NoSQL DBMS performance evaluation method.

References

[1] On the implementation of Commission Recommendation of 6 May 2003 concerning the definition of micro, small and medium-sized enterprises. Commission staff working document. [WWW]. European commission. 2009 http://ec.europa.eu/enterprise/policies/sme/files/sme_definition/sme_report_2009_en.pdf

[2] 2014 SBA Fact Sheet. European commission. 2014. [WWW] http://ec.europa.eu/enterprise/policies/sme/facts-figures-analysis/performance-review/files/countries-sheets/2014/estonia_en.pdf

[3] Kaarna, R., Masso, M., Rell, M. Väikese ja keskmise suurusega ettevõtete arengusuundumused. PRAXIS, 2012. [WWW] http://www.arengufond.ee/upload/Editor/ettevotlus/VKE_arengusuundumused_uuring_2012%20praxis.PDF

[4] Tiigiste, J. e-kaubandus on viimasel kümnendil hoogsalt arenenud. Statistikaamet. 2012 [WWW] https://statistikaamet.wordpress.com/2012/05/16/e-kaubandus-on-viimasel-kumnendil-hoogsalt-arenenud/

[5] Saaty, T. Theory an Application of the Analytic Network Process: Decision Making with Benefits, Opportunities, Costs, and Risks, , Pittsburgh: RWS Publications, 2005

[6] Turbyfill, C., Orji, C., Bitton, D. AS^{3}AP - An ANSI SQL Standard Scalable and Portable Benchmark for Relational Database Systems. [WWW] http://research.microsoft.com/en-us/um/people/gray/benchmarkhandbook/chapter5.pdf

[7] The Open Source Database Benchmark: What is OSDB? (FAQ) [WWW] http://osdb.sourceforge.net/index.php?page=faq

[8] Borrie, H. The Firebird Book: A Reference for Database Developers, Apress, 2004

[9] Worsey, J., Drake, J. Practical PostgreSQL. Sebastopol: O'Reilly, 2002.

[10] Catak, F.O., Karabas, S., Yildirim, S. Fuzzy Analytic Hierarchy Based DBMS Selection in Turkish National Identity Card Management Project. [WWW] http://airccse.org/journal/IS/papers/2412ijist03.pdf

[11] PAM authentication plugin. [WWW] https://mariadb.com/kb/en/mariadb/documentation/plugins/pam-authentication-plugin/

[12] FirebirdSQL: A not-so-very technical discussion of Multi Version Concurrency Control [WWW] http://www.firebirdsql.org/en/multi-version-concurrency-control/

[13] Tahriri, F., Dabbagh, M., Ebrahim N.A., Supplier Assessment and Selection Using Fuzzy Analytic Hierarchy Process in a Steel Manufacturing Company, Journal of Scientific Research & Reports, 2014. [WWW] http://zenodo.org/record/8612/files/Supplier_Assessment_-Nader_Ale_Ebrahim-_Tahriri_3102013JSRR8627.pdf

[14] Intel® Core™2 Extreme Processor X6800 and Intel® Core™2 Duo Desktop Processor E6000 and E4000 Sequences. Intel Corporation. October 2007. [WWW] http://download.intel.com/design/processor/datashts/31327807.pdf

[15] Haas, R., Meixner, O. An Illustrated Guide to the Analytic Hierarchy Process. Institute of Marketing & Innovation. University of Natural Resources and Applied Life Sciences, Vienna. [WWW] https://mi.boku.ac.at/ahp/ahptutorial.pdf

[16] MariaDB Documentation. SSL Overview. [WWW] https://mariadb.com/kb/en/mariadb/documentation/user-account-management/ssl-connections/ssl-overview/

[17] The Firebird FAQ. How to protect the connection over insecure networks (Internet)? [WWW] http://www.firebirdfaq.org/faq113/

[18] Evdoridis, T., Tzouramanis, T. A Generalized Comparison of Open Source and Commercial Database Management Systems. Chapter XXIII in St.Amant, K., Still, B. Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives. Information Science Reference. PA: IGI Global. New York, 2007

[19] Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, J., Lentz, A., Balling, D. High Performance MySQL, Second Edition. O'Reilly Media, Sebastopol, 2008

[20] Percona XtraBackup. [WWW] http://www.percona.com/software/percona-xtrabackup

[21] Create Trigger. PostgreSQL 9.2.9 Documentation. [WWW] http://www.postgresql.org/docs/9.2/static/sql-createtrigger.html

[22] Chapter 38: Procedural Languages. PostgreSQL 9.2.9 Documentation. [WWW] http://www.postgresql.org/docs/9.2/static/xplang.html

[23] H.3. Procedural Languages: Table H-2. Externally Maintained Procedural Languages. PostgreSQL 9.2.9 Documentation [WWW] http://www.postgresql.org/docs/9.2/static/external-pl.html

[24] Trigger Limitations. MariaDB Documentation. [WWW] https://mariadb.com/kb/en/mariadb/documentation/stored-programs-and-views/triggers/trigger-limitations/

[25] Using Triggers. 19.3. MySQL 5.7 Reference Manual. [WWW] http://dev.mysql.com/doc/refman/5.7/en/triggers.html

[26] Server-side programming. Firebird Documentation. [WWW] http://www.firebirdsql.org/manual/ufb-cs-serverprog.html

[27] Trigger. DDL statements. Firebird Documentation. [WWW] http://www.firebirdsql.org/refdocs/langrefupd21-ddl-trigger.html

[28] Writing UDFs for InterBase. Third-party Docs and Articles. Firebird Documentation. [WWW] http://www.firebirdsql.org/en/writing-udfs-for-interbase/

[29] Using Stored Routines (Procedures and Functions). 19.2. MySQL 5.7 Reference Manual. [WWW] http://dev.mysql.com/doc/refman/5.7/en/stored-routines.html

[30] Perl stored procedures for MariaDB. Percona Live. https://www.percona.com/live/mysql-conference-2013/sessions/perl-stored-procedures-mariadb

[31] Performance evaluation of MariaDB 10.1 and MySQL 5.7.4-labs-tplc. The MariaDB Blog. [WWW] https://blog.mariadb.org/performance-evaluation-of-mariadb-10-1-and-mysql-5-7-4-labs-tplc/

[32] Henschen, D. Oracle MySQL Upgrade Challenges NoSQL Onslaught. InformationWeek. [WWW] http://www.informationweek.com/software/information-management/oracle-mysql-upgrade-challenges-nosql-onslaught/d/d-id/1108523 (5.02.2013)

[33] PostgreSQL 9.2 released. PostgreSQL News. [WWW] http://www.postgresql.org/about/news/1415/ (10.09.2012)

[34] Streaming Replication. PostgreSQL Wiki. [WWW] https://wiki.postgresql.org/wiki/Streaming_Replication

[35] Multi-source replication. MariaDB Documentation. [WWW] https://mariadb.com/kb/en/mariadb/documentation/replication/standard-replication/multi-source-replication/

[36] Яковлев, С. MySQL и PostgreSQL. Часть 1. Сравнительный анализ. [WWW] http://www.ibm.com/developerworks/ru/library/os-mysql-postgresql/01/ (27.07.2010)

[37] Multi-source replication. MariaDB Documentation. [WWW] https://mariadb.com/kb/en/mariadb/documentation/replication/standard-replication/multi-source-replication/

[38] Postgres-XC Wiki. [WWW] http://postgresxc.wikia.com/wiki/Postgres-XC_Wiki

[39] How to do replication of Firebird databases? The Firebird FAQ [WWW] http://www.firebirdfaq.org/faq249/

[40] What is SymmetricDS. Overview. SymmetricDS official webpage. [WWW] http://www.symmetricds.org/about/overview

[41] Does Firebird support SMP? The Firebird FAQ. [WWW] http://www.firebirdfaq.org/faq2/

[42] Firebird – SuperServer, ClassicServer or SuperClassic? Sinática Information Technology Blog. [WWW] http://www.sinatica.com/blog/en/index.php/articles/firebird-superserver-classicserver-or-superclassic

[43] Limits on InnoDB Tables. 14.2.13. MySQL 5.0 Reference Manual. [WWW] http://dev.mysql.com/doc/refman/5.0/en/innodb-restrictions.html

[44] About. The official site for PostgreSQL. [WWW] http://www.postgresql.org/about/

[45] Database Limits. Firebird Technical Specifications. [WWW] http://www.firebirdsql.org/en/firebird-technical-specifications/

[46] pg_dump. PostgreSQL 9.2.9 Documentation. [WWW] http://www.postgresql.org/docs/9.2/static/app-pgdump.html

[47] mysqldump. MariaDB Documentation. [WWW] https://mariadb.com/kb/en/mariadb/documentation/clients-and-utilities/backup-restore-and-import/mysqldump/

[48] Why not use firebird? Stack Overflow. [WWW] http://stackoverflow.com/questions/1398491/why-not-use-firebird

[49] FBExport. Tools for Firebird developers. [WWW] http://fbexport.sourceforge.net/fbexport.php

[50] Firebird to MySQL. DBConvert. [WWW] https://dbconvert.com/convert-firebird-to-mysql-pro.php

[51] Converting MySQL to PostgreSQL. Wikibooks. [WWW] http://en.wikibooks.org/wiki/Converting_MySQL_to_PostgreSQL

[52] De la Cruz, S. How-To: Migrate PostgreSQL databases to MySQL using the MySQL Workbench Migration Wizard. The MySQL Workbench Team Blog. [WWW] http://mysqlworkbench.org/2012/11/how-to-migrate-postgresql-databases-to-mysql-using-the-mysql-workbench-migration-wizard/ (21.11.2012)

[53] MariaDB versus MySQL - Features. Knowledge Base. MariaDB Corporation. [WWW] https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/

[54] Appendix A: Firebird server architectures. Firebird 2.5 Quick Start. Firebird Documentation Index. [WWW] http://www.firebirdsql.org/manual/qsg25-appx-architectures.html

[55] IBPP, a C++ Client Interface to Firebird Server. [WWW] http://www.ibpp.org/

[56] PostgreSQL DROP DATABASE. PostgreSQLTutorial.com. [WWW] http://www.postgresqltutorial.com/postgresql-drop-database/

[57] How to drop a PostgreSQL database if there are active connections to it? [WWW] http://stackoverflow.com/questions/5408156/how-to-drop-a-postgresql-database-if-there-are-active-connections-to-it

[58] Escaped List Separator. Boost C++ libraries. [WWW] http://www.boost.org/doc/libs/1_36_0/libs/tokenizer/escaped_list_separator.htm

[59] What is FlameRobin? FlameRobin.org. [WWW] http://www.flamerobin.org/index.php

[60] pgAdmin. PostgreSQL Tools. [WWW] http://www.pgadmin.org/

[61] About. phpMyAdmin. [WWW] http://www.phpmyadmin.net/home_page/index.php

[62] What is phpPgAdmin? [WWW] http://phppgadmin.sourceforge.net/doku.php

[63] IBSurgeon Enterprise Pack. IBSurgeon. [WWW] http://www.ib-aid.com/

[64] What is FB TraceManager. Upscene. [WWW] http://www.upscene.com/fb_tracemanager/

[65] Features and Benefits. Sinática Monitor. Sinática. [WWW] http://www.sinatica.com/index.php/en/monitor

[66] Easy-IP Firebird Database Manager. Publisher's description. [WWW] http://easy-ip-firebird-database-manager.software.informer.com/

[67] Community Guide to PostgreSQL GUI Tools. PostgreSQL Wiki. [WWW] https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools

[68] pgFouine - a PostgreSQL log analyzer. [WWW] http://pgfouine.projects.pgfoundry.org/

[69] Performance Optimization. PostgreSQL Wiki. [WWW] https://wiki.postgresql.org/wiki/Performance_Optimization

[70] Monitoring. PostgreSQL Wiki. [WWW] https://wiki.postgresql.org/wiki/Monitoring

[71] PostgreSQL Performance Management. EnterpriseDB. [WWW] http://www.enterprisedb.com/solutions/postgresql-performance-management

[72] PostgreSQL Performance Tuning. PostgreSQL. Revolution Systems. [WWW] http://www.revsys.com/services/postgresql/tuning/

[73] The innotop MySQL and InnoDB monitor. Xaprb. Baron Schwartz Blog. [WWW] http://www.xaprb.com/blog/2006/07/02/innotop-mysql-innodb-monitor/

[74] Percona Toolkit for MySQL. Percona. [WWW] http://www.percona.com/software/percona-toolkit

[75] MySQLTuner-perl. [WWW] http://mysqltuner.com/

[76] Overview. MySQL Tuning Primer Script. [WWW] https://launchpad.net/mysql-tuning-primer

[77] Automate MySQL Development Tasks. Toad for MySQL. Dell Software. [WWW] http://www.quest.com/toad-for-mysql/

[78] Navicat for MySQL. PremiumSoft. [WWW] http://navicat.com/products/navicat-for-mysql

[79] Reference Manuals. Firebird. [WWW] http://www.firebirdsql.org/en/reference-manuals/

[80] PostgreSQL 9.2.9 Documentation. The PostgreSQL Global Development Group. [WWW] http://www.postgresql.org/docs/9.2/static/

[81] MariaDB Documentation. Knowledge Base. MariaDB Corporation. [WWW] https://mariadb.com/kb/en/mariadb/documentation/

[82] MySQL Documentation: MySQL Reference Manuals. MySQL.com. [WWW] http://dev.mysql.com/doc/

[83] DB-Engines Ranking. DB-Engines. [WWW] http://db-engines.com/en/ranking

[84] DB-Engines Ranking of Relational DBMS. DB-Engines. [WWW] http://db-engines.com/en/ranking/relational+dbms

[85] Percona MySQL performance blog. [WWW] http://www.percona.com/blog/search/mariadb/

[86] Effective MySQL. Ronald Bradford's site. [WWW] http://effectivemysql.com/

[87] Саати, Т. Принятие решений. Метод анализа иерархий. Перевод с английского Р. Г. Вачнадзе. Радио и связь. Москва. 1993.

[88] Description. Maatkit - Essential command-line utilities for MySQL. [WWW] http://www.maatkit.org/doc/maatkit.html

[89] Connolly, T., Begg., C. Database Systems. A Practical Approach to Design, Implementation, and Management. Fourth Edition. University Of Paisley, Pearson Education Limited, Essex, 2005

Appendices

Appendix 1 – OSDB benchmark queries

Benchmark name	SQL query
agg_create_view	CREATE VIEW REPORTVIEW (COL_KEY,COL_SIGNED,COL_DATE,COL_DECIM, COL_NAME,COL_CODE,COL_INT) AS SELECT UPDATES.COL_KEY, UPDATES.COL_SIGNED, UPDATES.COL_DATE, UPDATES.COL_DECIM, HUNDRED.COL_NAME, HUNDRED.COL_CODE, HUNDRED.COL_INT FROM UPDATES, HUNDRED WHERE UPDATES.COL_KEY = HUNDRED.COL_KEY;
agg_func	SELECT MIN(COL_KEY) FROM HUNDRED GROUP BY COL_NAME;
agg_info_retrieval	SELECT COUNT(COL_KEY) FROM TENPCT WHERE COL_NAME = 'THE ASAP BENCHMARKS' AND COL_INT <= 100000000 AND COL_SIGNED BETWEEN 1 AND 99999999 AND NOT (COL_FLOAT BETWEEN -450000000 AND 450000000) AND COL_DOUBLE > 600000000 AND COL_DECIM < -600000000";
agg_scal	SELECT MIN(COL_KEY) FROM UNIQUES;
agg_subtotal_report	SELECT AVG(COL_SIGNED), MIN(COL_SIGNED), MAX(COL_SIGNED), MAX(COL_DATE), MIN(COL_DATE), COUNT(DISTINCT COL_NAME), COUNT(COL_NAME), COL_CODE, COL_INT FROM REPORTVIEW WHERE COL_DECIM >980000000 GROUP BY COL_CODE, COL_INT;
agg_total_report	SELECT AVG(COL_SIGNED), MIN(COL_SIGNED), MAX(COL_SIGNED), MAX(COL_DATE), MIN(COL_DATE), COUNT(DISTINCT COL_NAME), COUNT(COL_NAME), COUNT(COL_CODE), COUNT(COL_INT) FROM REPORTVIEW WHERE COL_DECIM > 980000000;
bulk_append	INSERT INTO UPDATES SELECT * FROM SAVEUPDATES;
bulk_delete	DELETE FROM UPDATES WHERE COL_KEY < 0;
bulk_modify	UPDATE UPDATES SET COL_KEY = COL_KEY - 100000 WHERE COL_KEY BETWEEN 5000 AND 5999;
bulk_save	SELECT MIN(COL_KEY) FROM HUNDRED GROUP BY COL_NAME;
count_tuples	SELECT MIN(COL_KEY) FROM HUNDRED GROUP BY COL_NAME;

Benchmark name	SQL query
create_idx_hundred_code_h	MariaDB: CREATE INDEX HUNDRED_CODE_H ON HUNDRED(COL_CODE) USING HASH; PostgreSQL: CREATE INDEX HUNDRED_CODE_H ON HUNDRED USING HASH (COL_CODE);
create_idx_tenpct_code_h	MariaDB: CREATE INDEX TENPCT_CODE_H USING HASH ON TENPCT(COL_CODE); PostgreSQL: CREATE INDEX TENPCT_CODE_H ON TENPCT USING HASH (COL_CODE);
create_idx_tenpct_name_h	MariaDB: CREATE INDEX TENPCT_NAME_H ON TENPCT(COL_NAME) USING HASH; PostgreSQL: CREATE INDEX TENPCT_NAME_H ON TENPCT USING HASH (COL_NAME);
create_idx_uniques_code_h	MariaDB: CREATE INDEX UNIQUES_CODE_H ON UNIQUES(COL_CODE) USING HASH; PostgreSQL: CREATE INDEX UNIQUES_CODE_H ON UNIQUES USING HASH (COL_CODE);
create_idx_updates_code_h	MariaDB: CREATE INDEX UPDATES_CODE_H ON UPDATES(COL_CODE) USING HASH; PostgreSQL: CREATE INDEX UPDATES_CODE_H ON UPDATES USING HASH (COL_CODE);
join_2	SELECT UNIQUES.COL_SIGNED, UNIQUES.COL_NAME, HUNDRED.COL_SIGNED, HUNDRED.COL_NAME FROM UNIQUES, HUNDRED WHERE UNIQUES.COL_ADDRESS = HUNDRED.COL_ADDRESS AND UNIQUES.COL_ADDRESS = 'SILICON VALLEY';
join_2_cl	SELECT UNIQUES.COL_SIGNED, UNIQUES.COL_NAME, HUNDRED.COL_SIGNED, HUNDRED.COL_NAME FROM UNIQUES, HUNDRED WHERE UNIQUES.COL_KEY = HUNDRED.COL_KEY AND UNIQUES.COL_KEY =1000";
join_3_cl	SELECT UNIQUES.COL_SIGNED, UNIQUES.COL_DATE, HUNDRED.COL_SIGNED, HUNDRED.COL_DATE, TENPCT.COL_SIGNED, TENPCT.COL_DATE FROM UNIQUES, HUNDRED, TENPCT WHERE UNIQUES.COL_KEY = HUNDRED.COL_KEY AND UNIQUES.COL_KEY = TENPCT.COL_KEY AND UNIQUES.COL_KEY = 1000;
join_3_ncl	SELECT UNIQUES.COL_SIGNED, UNIQUES.COL_DATE, HUNDRED.COL_SIGNED, HUNDRED.COL_DATE, TENPCT.COL_SIGNED, TENPCT.COL_DATE FROM UNIQUES, HUNDRED, TENPCT WHERE UNIQUES.COL_CODE = HUNDRED.COL_CODE AND UNIQUES.COL_CODE = TENPCT.COL_CODE AND UNIQUES.COL_CODE = 'BENCHMARKS';

Benchmark name	SQL query
join_4_ncl	SELECT UNIQUES.COL_DATE, HUNDRED.COL_DATE, TENPCT.COL_DATE, UPDATES.COL_DATE FROM UNIQUES, HUNDRED, TENPCT, UPDATES WHERE UNIQUES.COL_CODE = HUNDRED.COL_CODE AND UNIQUES.COL_CODE = TENPCT.COL_CODE AND UNIQUES.COL_CODE = UPDATES.COL_CODE AND UNIQUES.COL_CODE = 'BENCHMARKS';
proj_100	SELECT DISTINCT COL_ADDRESS, COL_SIGNED FROM HUNDRED;
proj_10_pct	SELECT DISTINCT COL_SIGNED FROM TENPCT;
sel_100_cl	SELECT COL_KEY, COL_INT, COL_SIGNED, COL_CODE, COL_DOUBLE, COL_NAME FROM UPDATES WHERE COL_KEY <= 100;
sel_100_ncl	SELECT COL_KEY, COL_INT, COL_SIGNED, COL_CODE, COL_DOUBLE, COL_NAME FROM UPDATES WHERE COL_KEY <= 100;
sel_10pct_ncl	SELECT DISTINCT COL_SIGNED FROM TENPCT;
sel_1_cl	SELECT COL_KEY, COL_INT, COL_SIGNED, COL_CODE, COL_DOUBLE, COL_NAME FROM UPDATES WHERE COL_KEY = 1000;
sel_variable_select_high	SELECT COL_KEY, COL_INT, COL_SIGNED, COL_CODE, COL_DOUBLE, COL_NAME FROM TENPCT WHERE COL_SIGNED < -250000000;
sel_variable_select_low	SELECT COL_KEY, COL_INT, COL_SIGNED, COL_CODE, COL_DOUBLE, COL_NAME FROM TENPCT WHERE COL_SIGNED < -500000000;
table_scan	SELECT * FROM UNIQUES WHERE COL_INT = 1;
upd_append_duplicate	INSERT INTO UPDATES (COL_KEY, COL_INT, COL_SIGNED, COL_FLOAT,COL_DOUBLE, COL_DECIM, COL_DATE, COL_CODE, COL_NAME, COL_ADDRESS) VALUES(6000, 0, 60000, 39997.90, 50005.00, 50005.00, '11/10/1985', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO')
upd_app_t_end	INSERT INTO UPDATES VALUES (1000000001, 50005, 50005, 50005.00, 50005.00, 50005.00, '1/1/1988', 'CONTROLLER', 'ALICE IN WONDERLAND', 'UNIVERSITY OF ILLINOIS AT CHICAGO');
upd_app_t_mid	INSERT INTO UPDATES (COL_KEY, COL_INT, COL_SIGNED, COL_FLOAT, COL_DOUBLE, COL_DECIM, COL_DATE, COL_CODE, COL_NAME, COL_ADDRESS) VALUES (5005, 5005, 50005, 50005.00, 50005.00,

Benchmark name	SQL query
	50005.00,'1/1/1988', 'CONTROLLER', 'ALICE IN WONDERLAND','UNIVERSITY OF ILLINOIS AT CHICAGO');
upd_del_t_end	DELETE FROM UPDATES WHERE COL_KEY = '-1000';
upd_del_t_mid	DELETE FROM UPDATES WHERE (COL_KEY='5005') OR (COL_KEY='-5000');
upd_integrity_test	UPDATE HUNDRED SET COL_SIGNED = '-500000000' WHERE COL_INT = 0;
upd_mod_t_cod	UPDATE UPDATES SET COL_CODE = 'SQL+GROUPS' WHERE COL_KEY = 5005;
upd_mod_t_end	UPDATE UPDATES SET COL_KEY = '-1000' WHERE COL_KEY = 1000000001;
upd_mod_t_int	UPDATE UPDATES SET COL_INT = 50015 WHERE COL_KEY = 5005;
upd_mod_t_mid	UPDATE UPDATES SET COL_KEY = '-5000' WHERE COL_KEY = 5005;
upd_remove_duplicate	DELETE FROM UPDATES WHERE COL_KEY = 6000 AND COL_INT = 0;

Appendix 2 – Architecture of the experimental platform

The information of tested system hardware is presented below on Table 1.

Table 1 of Appendix 2. HW/SW components of tested system

Component	**Description**
System	Lenovo ThinkCentre M57 6069Y19
CPU	Intel(R) Core(TM)2 Duo CPU E6550 @ 2.33GHz
RAM	2 x 1 GB DDR2 @ 667 MHz
HDD	ST3320820AS 298 GB IDE
OS	OpenSuse 13.1
Kernel	3.11.10-21-desktop x86_64 Linux/GNU

The system is rather representative of desktop type computer than server, but all selection of this particular computer is justified at least by number of reason. First, the development of the benchmark tests is more convenient on the same platform where these tests run. Second, this was the only available computer system at the moment. At last, OpenSuse distributive was selected because this is most full, stable and user-friendly distributive. The OpenSuse provides many of RPM packages ready-from-box and much more can be added and reconfigured with YaST Control Center. All three versions of DBMSs (Table 12.) are installed from RPM packages in YaST.

Table 2 of Appendix 2. Versions of tested DBMS.

DBMS	**Version**
PostgreSQL	9.2.7. on x86_64-suse-linux-gnu, compiled by gcc (SUSE Linux) 4.8.1.(gcc-4_8-branch revision 202388) 64-bit
MariaDB	5.5.33-2.2-x86_64 OpenSuse package
Firebird SQL	SuperClassic version 2.5.2.26539-79.2

MariaDB is successful fork of MySQL that provides state-of-the-art features of the MySQL family like Percona-XtraDB storage engine [53] that replaces older InnoDB belonging to Oracle Inc. OpenSuse has switched to MariaDB since its version 12.3 like some other developers of Linux distributives to avoid risk of Oracle can close MySQL project. Nevertheless, in the experimental evaluation process have been out-off-box configuration settings and default storage engine (InnoDB) to provide fair competition between three DBMSs. Configuration files of all three DBMSs reside on disk attached to this work.

MariaDB has been shipped with in RPM packages with mysql CLI administration utility and separate phpMyAdmin web administration utility. PostgreSQL 9.2.7. installed on the platform on the installation moment (September 2014) was the last version provided with OpenSuse. This DBMS has psql CLI administration utility, and two separately installed GUI pgadmin3 and web administration utility phpPgAdmin. FirebirdSQL could be installed as one of four different available possible version: Classic Server, Superserver, SuperClassic and Embedded [54], besides there is only one architecture, excluding Embedded, can be installed at time on the platform. In this work has been installed and used SuperClassic architecture than was firstly introduced with FirebirdSQL 2.5. version. The decision of selection this very architecture type is based on the fact that SuperClassic version is trade-off between two polar Classic Server and Superserver models of server architecture. SuperClassic combines the full support of SMP as Classic Server does with possibility to use Superserver efficiency in deal of accommodation of possible growth of connection number [54].

The OSDB project that is available for download [55] contains only the code that after light modification is able to execute benchmarks only with MySQL. As consequence, the code has been adopted to work with two others DBMS – PostgreSQL and FirebirdSQL Server, therefore methods used for connection and database administration are different. Moreover, even SQL queries syntax differs in these three DBMS. This fact supposed a modification of the major part of the code and this has been done in the work (Figure 1 of Annex 2).

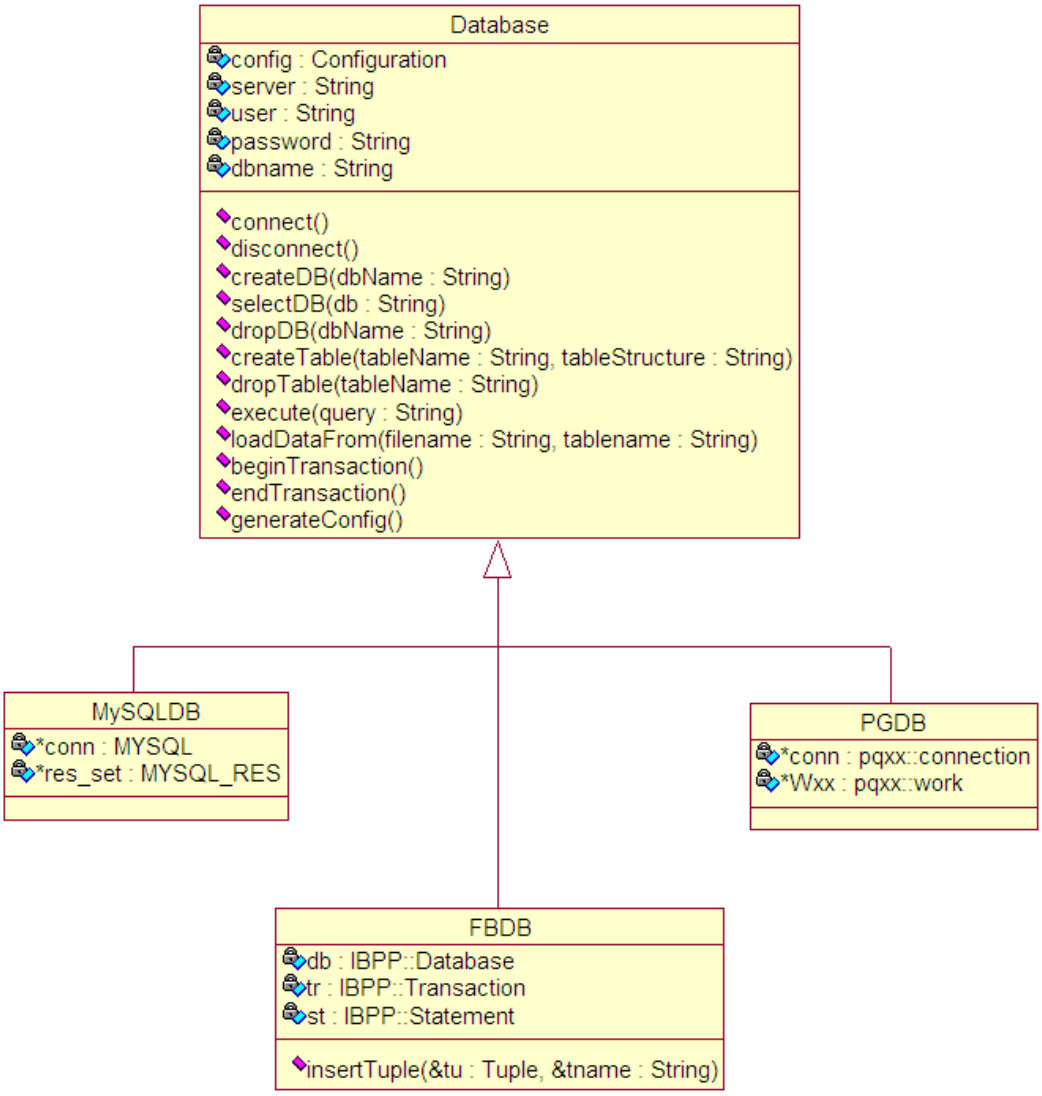

Figure 1 of Annex2. Implementation of Database class for different tested DBMS.

In this work were used C++ APIs :

1. pqxx (version 4.0.1-2.1.3-x86_64) for PostgreSQL
2. mysqlclient-devel (version 5.5.33-2.2-x86_64) for MariaDB
3. IBPP (version 2.5.3.1) for FirebirdSQL

Two first (pqxx and mysqlclient) were provided in OpenSuse RPM-package and the IBPP was downloaded, compiled and installed from sourceforge.net.

In order to be successfully compiled Makefile.am for different DBMSs should include next rows representing path to library:

1. AM_LDFLAGS=-L/usr/lib/mysql –lmysqlclient
2. AM_LDFLAGS=-L/usr/include/pqxx –lpqxx
3. AM_LDFLAGS=-L/usr/include/ibpp –libpp

The directories can vary in different distributions, therefore their locations should be verified. In all project has been used g++ (SUSE Linux) 4.8.1 (gcc-4_8-branch revision 202388) compiler.

The CSV file sets (40 or 4 MB) necessary for OSDB database populating can be downloaded from the project's page on sourceforge.net. The minor modifications have been also performed with downloaded osdb-0.90 project to be able to interact with MariaDB database. For example, string in original osdb CreateIdxUpdatesCodeH contains string "create index updates_code_h on updates HASH (col_code)" that is not compatible with current SQL supported by MariaDB and has been replaced with string "create index updates_code_h on updates (col_code) using hash". In addition, the connection string parameters have been hard coded to simplify testing.

Then the new osdbpg-0.91 project has been created on the base of OSDB for MySQL. Instead of mysqlclient library, the libpqxx has been linked to the project. In the subfolder „dbms"of the project has been added files with PGDB.h and PGDB.cpp. The files encapsulate the inner mechanism of the class PGDB with its members. The PGDB class is inherited from class Database. The instance of the PGDB class is used in main project OSDB class.

However, the PostgreSQL client library differs from MySQL van not only by signatures of the methods but in principles of these realizations. That was the reason to implement this class in relevance to client library. The instance of pqxx::work class has been added to the OSDB class. The work class encapsulates all operations with transactions in libpqxx library. The methods exec (String query) and commit () have been used to execute and commit. Besides, a database can be created in PostgreSQL only using nontransaction class of pqxx library.

```
void PGDB::createDB(string dbName)
{
	try{
		nontransaction nc(*conn, "empty");
		string c_string = "CREATE DATABASE " + dbName + " ENCODING 'UTF8' TEMPLATE
template1";
		result R = nc.exec(c_string);
		disconnect();
	}catch(const exception &e){
		cerr <<"Database "<<dbName<<" failed by reason : "<<e.what()<<endl;
	}
}
```

Figure 2 of Annex 2. Creating database using non-transaction class of libpqxx

The drop database procedure in PostgreSQL is even more complicated. As first step, it is required to revoke possibility to create new database connections using SQL query described in Figure 12. presented code snippet as string s1. Then, using s2 query string (Figure 12) the existing connection should be terminated [56, 57]. At last, the database can be dropped using SQL query “DROP DATABASE IF EXISTS”.

```
void PGDB::dropDB(string dbName)
{
    string s1 = "REVOKE CONNECT ON DATABASE "+dbname+" FROM PUBLIC, osdb, postgres;";
    string s2 = "SELECT pg_terminate_backend(pid) FROM pg_stat_activity WHERE pid <> pg_backend_pid()";
    string c_string = "DROP DATABASE IF EXISTS " + dbName;
    try
    {
        disconnect();
        connect();
        nontransaction nc(*conn, "empty");
        result R1 = nc.exec(s1);
        result R2 = nc.exec(s2);
        result R3 = nc.exec(c_string);
        cout << "DATABASE " << dbName << " DROPPED" << endl;
    }
    catch (const exception &e)
    {
        cerr << e.what() << endl;
    }
}
```

Figure 3 of Annex 3. Dropping database in PostgreSQL

For FirebirdSQL benchmarking has been created project osdbfb-0.91 that varies substantially from two previous OSDB benchmarking projects. The overall structure of the project has been remained the same, but significant modifications have been done for Database inherited class FBDB by the reason of lack of required for this project data import functionality discovered during osdbfb-0.91 project development. This functional mismatching has been thoroughly described in Chapter 2.4.5. Because relevant external data import is missing in FirebirdSQL the new data class Tuple has been added to the project.

```
class Tuple{
private:
	int count;
	std::string col_key;
	std::string col_int;
	std::string col_signed;
	std::string  col_float;
	std::string  col_double;
	std::string  col_decimal;
	std::string col_date;
	std::string col_code;
	std::string col_name;
	std::string col_address;

public:
	Tuple(void){ count = 0; }
	void getString(const std::string &str);
	int getCount(void);
	std::string getKey(void);
	std::string getInt(void);
	std::string getSigned(void);
	std::string getFloat(void);
	std::string getDouble(void);
	std::string getDecimal(void);
	std::string getDate(void);
	std::string getCode(void);
	std::string getName(void);
	std::string getAddress(void);
};

void Tuple::getString(const std::string &str)
{
	switch(count)
	{
		case 0:
			col_key = str;
			break;

		case 1:
			col_int = str;
			break;

		case 2:
			col_signed = str;
			break;

		case 3:
			col_float = str;
			break;

		case 4:
			col_double = str;
			break;

		case 5:
			col_decimal = str;
			break;

		case 6:
			col_date = str;
			break;
```

```
                case 7:
                        col_code = str;
                        break;

                case 8:
                        col_name = str;
                        break;

                case 9:
                        col_address = str;
                        break;
        }
        ++count;
}

int Tuple::getCount() { return count; }
std::string Tuple::getKey() { return col_key; }
std::string Tuple::getInt() { return col_int; }
std::string Tuple::getSigned() { return col_signed; }
std::string Tuple::getFloat() { return col_float; }
std::string Tuple::getDouble() { return col_double; }
std::string Tuple::getDecimal() { return col_decimal; }
std::string Tuple::getDate() { return col_date; }
std::string Tuple::getCode() { return col_code; }
std::string Tuple::getName() { return col_name; }
std::string Tuple::getAddress() { return col_address; }
```

Figure 4 of Appendix 2. Class Tuple with methods.

In addition, the new method insertTuple has been added to FBDB class. This method composes SQL query “INSERT INTO” from the instance of Tuple class and string variable containing the table name.

```
string FBDB::insertTuple(Tuple &tu, const std::string &tname)
{
    string  s =  "INSERT INTO "+tname+"(col_key, col_int,"
         " col_signed, col_float, col_double,"
         " col_decim, col_date, col_code,"
         " col_name, col_address)"
        " VALUES ("+ tu.getKey()+
        ", "+ tu.getInt()+", "
        + tu.getSigned() +", "
        + tu.getFloat()  +", "
        + tu.getDouble() +", "
        + tu.getDecimal() +", '"
        + tu.getDate() +"', '"
        + tu.getCode() +"', '"
        + tu.getName() +"', '"
        + tu.getAddress()
        +"');";
    return s;
}
```

Figure 5 of Appendix 2. Method insertTuple of FBDB class.

The method loadDataFrom that was inherited from Database has been drastically changed. Now this method implements escaped_list_separator class of boost C++ library [56] to read coma-separated data from file. It instantiates an object of Tuple class that latter is used in SQL composition with insertTuple method. The query is submitted to method “execute” wrapping interaction with transaction mechanism in all three projects. The header of boost library have to be included in FBDB header file.

```
void FBDB::loadDataFrom(string name)
{
    try
    {
        string line;
        ifstream myfile((PATH + name).c_str());
        beginTransaction();
        if (myfile.is_open())
        {
            while ( myfile.good() )
            {
                getline (myfile,line);
                Tuple tuple;
                tokenizer<escaped_list_separator<char> > tok(line);
                for(tokenizer<escaped_list_separator<char> >::iterator beg=tok.begin(); beg!=tok.end();++beg)
                {
                    string temp = *beg;
                    tuple.getString(temp);
                }
                st->Execute(insertTuple(tuple, name));
            }
            myfile.close();
        }
        else cout << "Unable to open file "<<PATH + name<<endl;
        endTransaction();
    }
    catch (const exception &e)
    {
        cerr << e.what() << endl;
    }
}
```

Figure 6 of Appendix 2. Implementation of loadDataFrom in FBDB class.

For power consumption and performance evaluation, it was necessary to know the exact number of performed transaction. The different numbers are based on the different functionality and realization. These numbers are calculated in Appendix 3.

The main file of OSDB project OSDB.cpp has been modified to run continuously during the one hour. This multiplier 52 is implemented for PostgreSQL, and for other project were used relevant numbers: 54 for MariaDB and 43 for FirebirdSQL

```
int main(int argc, char** argv)
{
	Timer tm;
	double elapsedTime = 0.0;
	int times = 0;
	tm.startTimer();
	while(elapsedTime < 3600.0)
	{
		OSDB thisInstance(argc, argv);
		thisInstance.runBenchmarks();
		elapsedTime =+ tm.stopTimer();
		times++;
		cout<<" "<<endl;
		cout<<"Elapsed time "<< elapsedTime<<endl;
		cout<<times*52<<" transactions performed"<<endl;
	}
	return 0;
}
```

Figure 7 of Appendix 2. Code snippet of OSDB.cpp file.

The idea of this approach is to evaluate number of transaction committed during the one hour to evaluate performance and power efficiency of every DBMS.

All program codes used in current work have been added to attached disk.

Appendix 3 – Number of transactions in test

Table 1 of Appendix 3. Number of transactions in test

Nr	Test name	PostgreSQL 9.2.7	MariaDB 5.5.33	FirebirdSQL 2.5.2
1	Database creation	0	1	0
2	Table creation	5	5	0
3	Dataset load	5	5	0
4	agg_create_view	1	1	1
5	agg_func	1	1	1
6	agg_info_retrieval	1	1	1
7	agg_scal	1	1	1
8	agg_subtotal_report	1	1	1
9	agg_total_report	1	1	1
10	bulk_append	1	1	1
11	bulk_delete	1	1	1
12	bulk_modify	1	1	1
13	bulk_save	1	1	1
14	count_tuples	1	1	1
15	create_idx_hundred_code_h	1	1	0
16	create_idx_tenpct_code_h	1	1	0
17	create_idx_tenpct_name_h	1	1	0
18	create_idx_uniques_code_h	1	1	0
19	create_idx_updates_code_h	1	1	0
20	join_2	1	1	1
21	join_2_cl	1	1	1
22	join_3_cl	1	1	1
23	join_3_ncl	1	1	1
24	agg_func	1	1	1
25	join_4_ncl	1	1	1
26	proj_100	1	1	1
27	proj_10_pct	1	1	1
28	sel_100_cl	1	1	1
29	sel_100_ncl	1	1	1

Nr	Test name	PostgreSQL 9.2.7	MariaDB 5.5.33	FirebirdSQL 2.5.2
30	sel_10pct_ncl	1	1	1
31	sel_1_cl	1	1	1
32	sel_variable_select_high	1	1	1
33	sel_variable_select_low	1	1	1
34	table_scan	1	1	1
35	upd_append_duplicate	1	1	1
36	upd_app_t_end	1	1	1
37	upd_app_t_mid	1	1	1
38	upd_del_t_end	1	1	1
39	upd_del_t_mid	1	1	1
40	upd_integrity_test	1	1	1
41	upd_mod_t_cod	1	1	1
42	upd_mod_t_end	1	1	1
43	upd_mod_t_int	1	1	1
44	upd_mod_t_mid	1	1	1
45	upd_remove_duplicate	1	1	1
46	drop_agg_view	0	0	1
47	Drop database	0	1	0
TTL		52	54	43

Appendix 4 – Performance evaluation results

Table 1 of Appendix 4. Results of performance benchmarking (seconds)

Test name	**PostgreSQL 9.2.7**	**MariaDB 5.5.33**	**FirebirdSQL 2.5.2**
Database creation	0.842518	0.00269389	0.185572
Table creation	0.0512509	0.888244	2.71767
Dataset load	2.48202	20.3042	448.251*
agg_create_view	0.00862002	0.143817	0.390566
agg_func	0.0666161	0.113545	0.0581758
agg_info_retrieval	0.034838	0.0867929	0.049927
agg_scal	0.034066	0.048306	0.0332489
agg_subtotal_report	0.214588	18.8833	0.0332561
agg_total_report	0.197711	18.8765	0.016845
bulk_append	0.478797	3.2737	failed
bulk_delete	0.04476	0.245332	0.0994558
bulk_modify	0.220317	0.242565	9.75764
bulk_save	0.0539119	0.112962	0.0169928
count_tuples	0.052794	0.110906	0.0165181
create_idx_hundred_code_h	0.177024	1.1546	N/A
create_idx_tenpct_code_h	0.183337	1.30896	N/A
create_idx_tenpct_name_h	0.533135	1.91246	N/A
create_idx_uniques_code_h	0.183253	0.749002	N/A
create_idx_updates_code_h	0.374942	1.86958	N/A
join_2	0.05656	0.131412	0.23788
join_2_cl	0.0199289	0.116559	0.182784
join_3_cl	0.0207541	0.119782	0.266508
join_3_ncl	0.000867128	0.000396013	0.291473
agg_func	0.0203559	0.111908	9.41249
join_4_ncl	0.000828981	0.00039506	0.374785
proj_100	4.09898	0.140239	0.254043

Test name	PostgreSQL 9.2.7	MariaDB 5.5.33	FirebirdSQL 2.5.2
proj_10_pct	0.11451	0.116066	0.138842
sel_100_cl	0.0656519	0.153201	0.0153391
sel_100_ncl	0.043731	0.151931	0.0164399
sel_10pct_ncl	0.000550032	0.000246048	0.0166061
sel_1_cl	0.0375061	0.149212	0.0165019
sel_variable_select_high	0.0839369	0.109101	0.01653
sel_variable_select_low	0.0221109	0.075038	0.0166512
table_scan	0.0216441	0.104612	0.0165169
upd_append_duplicate	0.0233419	0.000787973	0.132842
upd_app_t_end	0.00815701	7.82013e-05	0.116557
upd_app_t_mid	0.00829101	0.00012207	0.199832
upd_del_t_end	0.037215	0.22138	0.116531
upd_del_t_mid	0.0543849	0.228939	0.133225
upd_integrity_test	0.0250139	0.1102	0.0998979
upd_mod_t_cod	0.0381589	0.222935	0.10823
upd_mod_t_end	0.0449889	0.221459	0.0999041
upd_mod_t_int	0.038198	0.222496	0.108262
upd_mod_t_mid	0.0378802	0.222431	0.0998709
upd_remove_duplicate	0.0403731	0.222935	0.099906

* - Low performance caused by custom C++ implementation due to lack of native data upload methods.

Table 2 of Appendix 4. Results deviation from average value (seconds)

Test name	**Average**	**PostgreSQL 9.2.7**	**MariaDB 5.5.33**	**FirebirdSQL 2.5.2**
Database creation	0.3435946	0.4989234*	-0.3409007*	-0.1580226
Table creation	1.2190550	-1.1678041	-0.3308110	1.4986150
Dataset load	157.01241	-154.53039	-136.70821	291.23859
agg_create_view	0.1810010	-0.1723810	-0.0371840	0.2095650
agg_func	0.0794456	-0.0128295	0.0340994	-0.0212698
agg_info_retrieval	0.0571860	-0.0223480	0.0296069	-0.0072590
agg_scal	0.0385403	-0.0044743	0.0097657	-0.0052914
agg_subtotal_report	6.3770480	-6.1624600	12.5062520	-6.3437919
agg_total_report	6.3636853	-6.1659743	12.5128147	-6.3468403
bulk_append	1.8762485	-1.3974515	1.3974515	N/A
bulk_delete	0.1298493	-0.0850893	0.1154827	-0.0303935
bulk_modify	3.4068407	-3.1865237	-3.1642757	6.3507993
bulk_save	0.0612889	-0.0073770	0.0516731	-0.0442961
count_tuples	0.0600727	-0.0072787	0.0508333	-0.0435546
create_idx_hundred_code_h	0.6658120	-0.4887880	0.4887880	N/A
create_idx_tenpct_code_h	0.7461485	-0.5628115	0.5628115	N/A
create_idx_tenpct_name_h	1.2227975	-0.6896625	0.6896625	N/A
create_idx_uniques_code_h	0.4661275	-0.2828745	0.2828745	N/A
create_idx_updates_code_h	1.1222610	-0.7473190	0.7473190	N/A
join_2	0.1419507	-0.0853907	-0.0105387	0.0959293
join_2_cl	0.1064240	-0.0864951	0.0101350	0.0763600
join_3_cl	0.1356814	-0.1149273	-0.0158994	0.1308266
join_3_ncl	0.0975787	-0.0967116	-0.0971827	0.1938943
agg_func	3.1815846	-3.1612287	-3.0696766	6.2309054

Test name	Average	PostgreSQL 9.2.7	MariaDB 5.5.33	FirebirdSQL 2.5.2
join_4_ncl	0.1253363	-0.1245074	-0.1249413	0.2494487
proj_100	1.4977540	2.6012260	-1.3575150	-1.2437110
proj_10_pct	0.1231393	-0.0086293	-0.0070733	0.0157027
sel_100_cl	0.0780640	-0.0124121	0.0751370	-0.0627249
sel_100_ncl	0.0707006	-0.0269696	0.0812304	-0.0542607
sel_10pct_ncl	0.0058007	-0.0052507	-0.0055547	0.0108054
sel_1_cl	0.0677400	-0.0302339	0.0814720	-0.0512381
sel_variable_select_high	0.0698560	0.0140809	0.0392450	-0.0533260
sel_variable_select_low	0.0379334	-0.0158225	0.0371046	-0.0212822
table_scan	0.0475910	-0.0259469	0.0570210	-0.0310741
upd_append_duplicate	0.0523240	-0.0289821	-0.0515360	0.0805180
upd_app_t_end	0.0415974	-0.0334404	-0.0415192	0.0749596
upd_app_t_mid	0.0694150	-0.0611240	-0.0692930	0.1304170
upd_del_t_end	0.1250420	-0.0878270	0.0963380	-0.0085110
upd_del_t_mid	0.1388496	-0.0844647	0.0900894	-0.0056246
upd_integrity_test	0.0783706	-0.0533567	0.0318294	0.0215273
upd_mod_t_cod	0.1231080	-0.0849491	0.0998270	-0.0148780
upd_mod_t_end	0.1221173	-0.0771284	0.0993417	-0.0222132
upd_mod_t_int	0.1229853	-0.0847873	0.0995107	-0.0147233
upd_mod_t_mid	0.1200607	-0.0821805	0.1023703	-0.0201898
upd_remove_duplicate	0.1210714	-0.0806983	0.1018636	-0.0211654

* - Red is the worst and green is the best case comparing with average.

Appendix 5 – Power efficiency evaluation results

```
Open Source Database Benchmark 2014 for PostgreSQL 9.2.7
Reading configuration file: osdb.conf
Connected to osdb database.
agg_create_view completed in 0.0465541 seconds
agg_func completed in 0.0534592 seconds
agg_info_retrieval completed in 0.00230503 seconds
agg_scal completed in 0.020071 seconds
agg_subtotal_report completed in 0.113561 seconds
agg_total_report completed in 0.0963051 seconds
bulk_delete completed in 0.0386801 seconds
bulk_modify completed in 0.043555 seconds
bulk_save completed in 0.0516081 seconds
count_tuples completed in 0.0515258 seconds
join_2 completed in 0.0517051 seconds
join_2_cl completed in 0.0181758 seconds
join_3_cl completed in 0.0181401 seconds
join_3_ncl completed in 0.000577927 seconds
agg_func completed in 0.018137 seconds
join_4_ncl completed in 0.000602961 seconds
proj_100 completed in 0.0680399 seconds
proj_10_pct completed in 0.145765 seconds
sel_100_cl completed in 0.036139 seconds
sel_100_ncl completed in 0.0352778 seconds
sel_10pct_ncl completed in 0.000323057 seconds
sel_1_cl completed in 0.0342 seconds
sel_variable_select_high completed in 0.0815051 seconds
sel_variable_select_low completed in 0.019217 seconds
table_scan completed in 0.0195658 seconds
upd_append_duplicate completed in 0.00457716 seconds
upd_app_t_end completed in 0.00827384 seconds
upd_app_t_mid completed in 0.00829601 seconds
upd_del_t_end completed in 0.0342679 seconds
upd_del_t_mid completed in 0.0489411 seconds
upd_integrity_test completed in 0.0249379 seconds
upd_mod_t_cod completed in 0.0365782 seconds
upd_mod_t_end completed in 0.0382822 seconds
upd_mod_t_int completed in 0.0362132 seconds
upd_mod_t_mid completed in 0.0352621 seconds
upd_remove_duplicate completed in 0.044971 seconds
drop_agg_view completed in 0.00827408 seconds

Elapsed time 3600.92 seconds
93573 transactions performed
disconnected
```

Figure 1 of Appendix 5. Results of PostgreSQL testing during one hour.

```
Open Source Database Benchmark 2014 for MariaDB 5.5.33
Reading configuration file: osdb.conf
Connected to osdb database.
agg_create_view completed in 0.0349932 seconds
agg_func completed in 0.0125868 seconds
agg_info_retrieval completed in 0.00911403 seconds
agg_scal completed in 0.00491714 seconds
agg_subtotal_report completed in 0.130498 seconds
agg_total_report completed in 0.130227 seconds
bulk_delete completed in 0.0100679 seconds
bulk_modify completed in 0.00995803 seconds
bulk_save completed in 0.0310879 seconds
count_tuples completed in 0.011852 seconds
join_2 completed in 0.013736 seconds
join_2_cl completed in 0.0123658 seconds
join_3_cl completed in 0.00634193 seconds
join_3_ncl completed in 0.0196779 seconds
agg_func completed in 0.0110629 seconds
join_4_ncl completed in 0.024786 seconds
proj_100 completed in 0.0149472 seconds
proj_10_pct completed in 0.0106549 seconds
sel_100_cl completed in 0.00731087 seconds
sel_100_ncl completed in 0.00700188 seconds
sel_10pct_ncl completed in 0.00800514 seconds
sel_1_cl completed in 0.00675488 seconds
sel_variable_select_high completed in 0.0111549 seconds
sel_variable_select_low completed in 0.00779605 seconds
table_scan completed in 0.010432 seconds
upd_append_duplicate completed in 0.000181198 seconds
upd_app_t_end completed in 7.00951e-05 seconds
upd_app_t_mid completed in 6.91414e-05 seconds
upd_del_t_end completed in 0.0101638 seconds
upd_del_t_mid completed in 0.010452 seconds
upd_integrity_test completed in 0.0113299 seconds
upd_mod_t_cod completed in 0.0102081 seconds
upd_mod_t_end completed in 0.0101962 seconds
upd_mod_t_int completed in 0.010124 seconds
upd_mod_t_mid completed in 0.010246 seconds
upd_remove_duplicate completed in 0.010201 seconds
drop_agg_view completed in 0.00648904 seconds

Elapsed time 3600.08 seconds
200466 transactions performed
```

Figure 2 of Appendix 5. Results of MariaDB testing.

```
Open Source Database Benchmark 2014 for FirebirdSQL 2.5.2.26539-79.2
Reading configuration file: osdb.conf
Connected
agg_create_view completed in 0.277426 seconds
agg_func completed in 0.00820398 seconds
agg_info_retrieval completed in 0.0167801 seconds
agg_scal completed in 0.00809908 seconds
agg_subtotal_report completed in 0.016763 seconds
agg_total_report completed in 0.00810409 seconds
bulk_delete completed in 0.108264 seconds
bulk_modify completed in 0.108229 seconds
bulk_save completed in 0.00825787 seconds
count_tuples completed in 0.00829816 seconds
join_2 completed in 0.225388 seconds
join_2_cl completed in 0.191138 seconds
join_3_cl completed in 0.258156 seconds
join_3_ncl completed in 0.28314 seconds
agg_func completed in 0.34148 seconds
join_4_ncl completed in 0.374769 seconds
proj_100 completed in 0.287355 seconds
proj_10_pct completed in 0.147315 seconds
sel_100_cl completed in 0.00646901 seconds
sel_100_ncl completed in 0.00829291 seconds
sel_10pct_ncl completed in 0.00830317 seconds
sel_1_cl completed in 0.00830412 seconds
sel_variable_select_high completed in 0.00830507 seconds
sel_variable_select_low completed in 0.00832105 seconds
table_scan completed in 0.00826597 seconds
upd_append_duplicate completed in 0.025053 seconds
upd_app_t_end completed in 0.033263 seconds
upd_app_t_mid completed in 0.033252 seconds
upd_del_t_end completed in 0.108152 seconds
upd_del_t_mid completed in 0.133214 seconds
upd_integrity_test completed in 0.174867 seconds
upd_mod_t_cod completed in 0.0999041 seconds
upd_mod_t_end completed in 0.0999 seconds
upd_mod_t_int completed in 0.108228 seconds
upd_mod_t_mid completed in 0.091568 seconds
upd_remove_duplicate completed in 0.0998988 seconds
drop_agg_view completed in 0.0999482 seconds

Elapsed time 3603.69 seconds
33041 transactions performed
```

Figure 3 of Appendix 5. Results of FirebirdSQL testing.

Table 1 of Appendix 5. The power consumption measurement results (W).

Elapsed time	**PostgreSQL**	**MariaDB**	**FirebirdSQL**	**Idle computer**
0:01:00	1.60634	1.64058	1.43726	1.09018
0:02:00	3.24322	3.30791	2.85761	2.17359
0:03:00	4.89388	4.97836	4.28994	3.26528
0:04:00	6.55664	6.66175	5.71459	4.34583
0:05:00	8.17039	8.27233	7.12347	5.43079
0:06:00	9.79653	9.9071	8.54891	6.51455
0:07:00	11.4138	11.6174	9.99042	7.59364
0:08:00	13.0414	13.3213	11.4266	8.68023
0:09:00	14.6742	15.0401	12.8491	9.75864
0:10:00	16.302	16.7416	14.2831	10.8387
0:11:00	17.9397	18.3736	15.7343	11.9255
0:12:00	19.5807	19.9935	17.1437	13.0147
0:13:00	21.2189	21.6251	18.5976	14.1032
0:14:00	22.859	23.2539	20.0379	15.1882
0:15:00	24.5011	24.8717	21.4488	16.2593
0:16:00	26.1271	26.5001	22.9037	17.337
0:17:00	27.7484	28.1303	24.324	18.4148
0:18:00	29.3769	29.8336	25.7701	19.4876
0:19:00	31.0196	31.556	27.2178	20.5585
0:20:00	32.6479	33.283	28.6377	21.6337
0:21:00	34.2889	35.0039	30.0894	22.7096
0:22:00	35.9201	36.6662	31.5239	23.7871
0:23:00	37.5536	38.2857	32.9512	24.8654
0:24:00	39.1871	39.9153	34.3768	25.9402
0:25:00	40.8109	41.5497	35.7881	27.0222
0:26:00	42.4464	43.1836	37.2118	28.0976
0:27:00	44.0812	44.8129	38.6418	29.1731
0:28:00	45.756	46.434	40.0721	30.2486
0:29:00	47.4451	48.0527	41.4787	31.319
0:30:00	49.1435	49.6758	42.8986	32.3969
0:31:00	50.8381	51.3004	44.3584	33.4743

Elapsed time	PostgreSQL	MariaDB	FirebirdSQL	Idle computer
0:32:00	52.5073	52.9259	45.7969	34.556
0:33:00	54.1641	54.5485	47.2194	35.6331
0:34:00	55.8124	56.1695	48.6351	36.7103
0:35:00	57.4616	57.786	50.0427	37.7875
0:36:00	59.1076	59.3845	51.5068	38.8623
0:37:00	60.7586	60.9881	52.933	39.9398
0:38:00	62.4032	62.5982	54.3759	41.0168
0:39:00	64.0539	64.2245	55.8099	42.0907
0:40:00	65.7051	65.8351	57.2593	43.1641
0:41:00	67.3476	67.4566	58.7267	44.2417
0:42:00	68.991	69.057	60.1573	45.3178
0:43:00	70.6276	70.7496	61.6007	46.3849
0:44:00	72.2728	72.4701	63.037	47.4643
0:45:00	73.9089	74.1989	64.5077	48.5359
0:46:00	75.5497	75.9204	65.9704	49.6105
0:47:00	77.1885	77.5761	67.3869	50.6864
0:48:00	78.8204	79.1969	68.8351	51.7675
0:49:00	80.4574	80.8165	70.2692	52.8397
0:50:00	82.0972	82.4331	71.6953	53.9138
0:51:00	83.7419	84.06	73.1235	54.9857
0:52:00	85.3758	85.6863	74.58	56.0555
0:53:00	87.0175	87.3128	76.0073	57.1451
0:54:00	88.6597	88.9494	77.4821	58.2243
0:55:00	90.3015	90.5782	78.9187	59.3013
0:56:00	91.9338	92.1994	80.3823	60.3792
0:57:00	93.5663	93.8194	81.8362	61.4513
0:58:00	95.1996	95.4364	83.294	62.5261
0:59:00	96.8329	97.0506	84.735	63.6035
1:00:00	98.469	98.6856	86.1914	64.6793

Appendix 6 – Temperature and CPU utilization

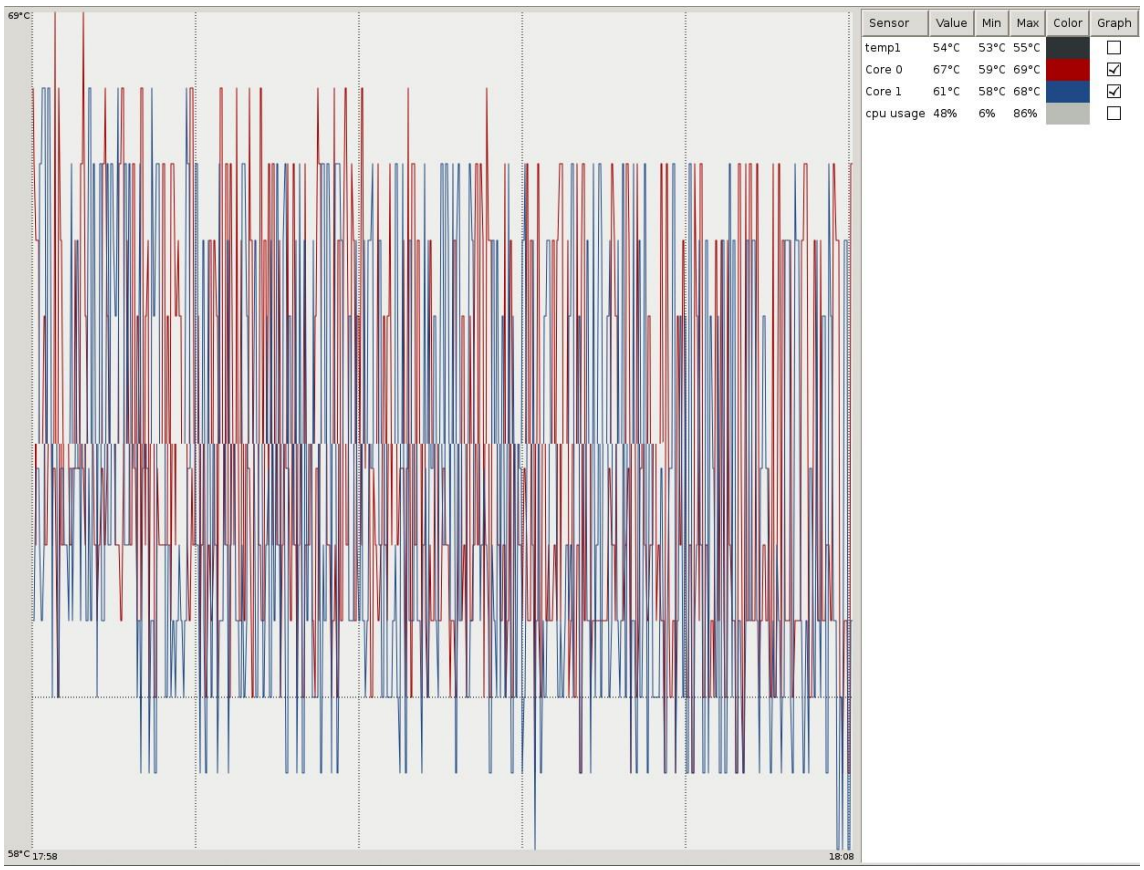

Figure 1 of Appendix 6. Temperature during tests of FirebirdSQL.

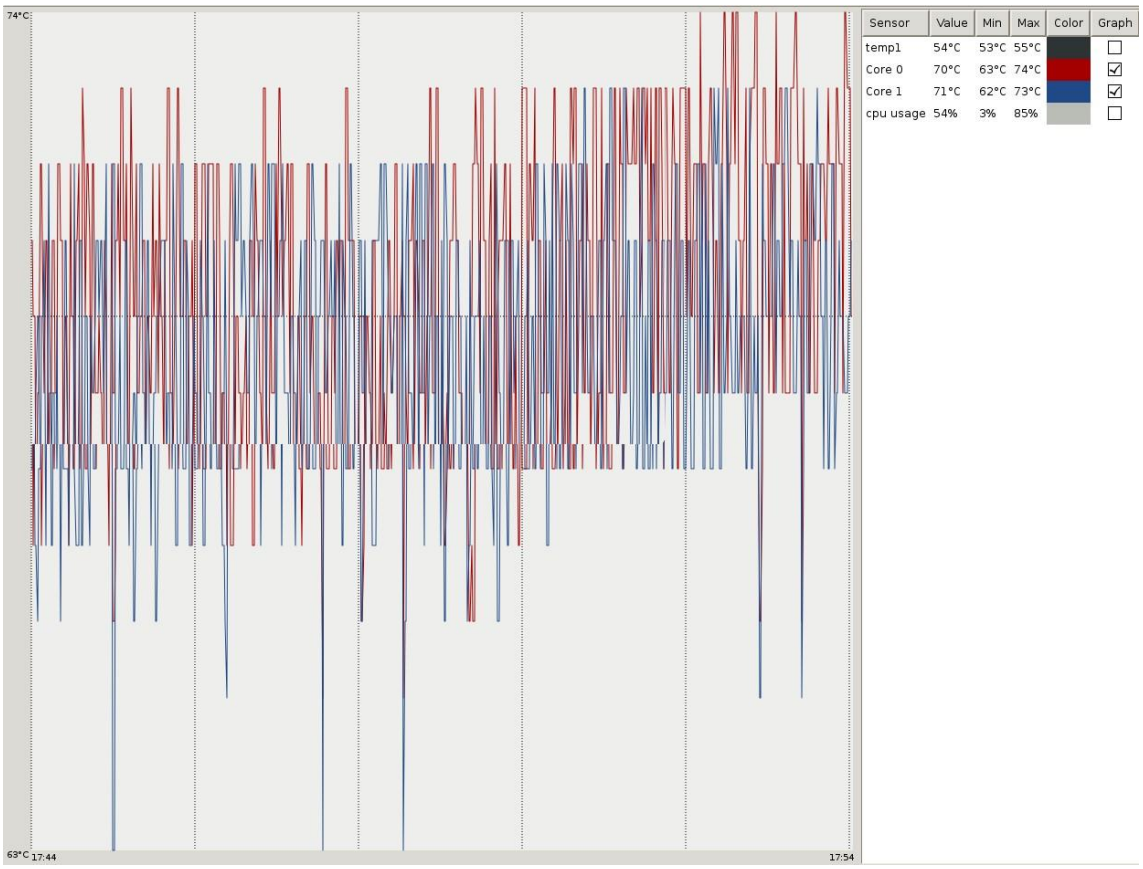

Figure 2 of Appendix 6. Temperature during tests of PostgreSQL.

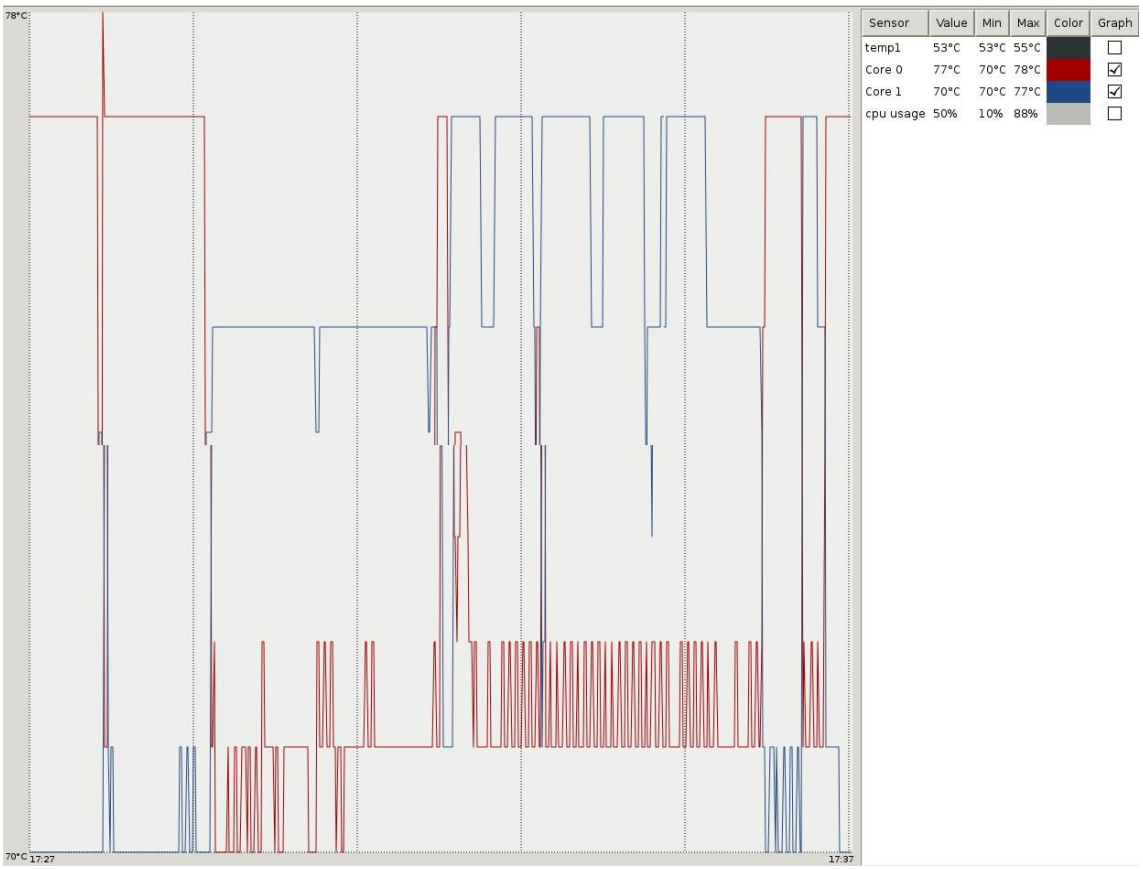

Figure 3 of Appendix 6. Temperature during tests of MariaDB.