Otto-von-Guericke-University of Magdeburg

Institute of Technical and Business Information Systems

Master's Thesis

Development of a Web-based Demonstrator for an Approach to prevent Insider Attacks on DBMS

Author:

Olexandr Shamin

Matriculation No. 199050

July 11, 2014

Advisors:

Dr. Ing. Eike Schallehn

M.Sc. Stefan Barthel

Databases and Software Engineering Workgroup

Shamin, Olexandr:
Development of a Web-based Demonstrator for an Approach to prevent Insider Attacks on DBMS
Master's Thesis, Otto-von-Guericke-University of Magdeburg, 2014.

Abstract

Every year the data leakage problem causes serious monetary as well as reputation losses for organizations. Adequate prevention and detection controls are challenging to define and implement, since no single type of control is universally effective, the defense in depth is required [McC08]. In this work, we address the particular attention to data leakage problems as well as MVAL approach to prevent them. This approach is proposed by Barthel and Schahllen [BS13b, BS13a] and is based on a calculation of monetary value of retrieving data from DBMS. For the purpose of studying MVAL approach, as well as presenting its work, we design and implement a web based demonstrator platform, which clearly exhibits the main properties of approach and can be used in the future as a core for further developing and extending functionality of MVAL approach.

Keywords: databases, security, insider threat, data leakage

Acknowledgements

This thesis would never have been completed successfully without the help of following people.

First and foremost, I would like to thanks my adviser M.Sc. Stefan Barthel, for his patient guidance during selecting of research topic as well as generous contribution of knowledge and valuable comments during the writing of the thesis.

I would also like to thanks a second adviser Dr. Eike Schallehn, who always promptly answered on all questions and helped with administration issues during the work.

Furthermore, I must also express gratitude to my family and girlfriend, who continuously supported me and were very patient for my limited time during this work.

Acronyms

3VL	Three Valued Logic
ACL	Access Control List
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interfaces
AST	Abstract Syntax Tree
CERT	Computer Emergency Response Team
CIA	Confidentiality, Integrity and Availability
CSS	Cascading Style Sheets
DAC	Discretionary Access Control
DB	Database
DBA	Database administrator
DBMS	Database Management System
DBS	Database system
DCL	Data control language
DDL	Data Definition Language
DES	Data Encryption Standard
DML	Data Manipulation Language
DOM	Document Object Model
ECMA	European Computer Manufacturers Association

GUI	Graphical User Interface
HTML	HyperText Markup Language
IEC	International Electrotechnical Commission
ISO	International Standards Organization
JSON	JavaScript Object Notation
LQP	Logical Query Plan
MAC	Mandatory Access Control
MVAL	Monetary Value
OS	Operating System
RAML	Relational Algebra Markup Language
RAT	Relational Algebra Toolkit
RATTAIL	Relational Algebra Toolkit Automated Instruction Language
RBAC	Role Based Access Control
RDB	Relational database
RDBMS	Relational Database Management System
RDML	Relational Database Markup Language
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TCL	Transaction Control Language
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Contents

List of Figures

List of Tables

List of Code Listings

List of Algorithms

1. Introduction

In the information age, computer systems have become more complicated as attacks on them [CIS14]. Therefore, preserving the security of sensitive data has become a major research concern. While traditionally information security measure were focused on defending systems against outside threat, threats coming from inside of organization are carrying more risk and damage. By the CERT 2013 US State of Cybercrime Survey [Uni13], among big and small organizations, 53 % of respondents reported that damage caused by insider attacks overtake damage of outsider attacks. Unintentional exposure of private or sensitive data, theft of intellectual property or similar proprietary information (such as customer, financial records) cause the most damage among all insider incidents and certainly gather the most sensational publicly.

Last huge leakage happened at the end of 2013 with Adobe Systems, multimedia and creativity software developing company. By the official report [Ark13], attackers stole from their systems information relating to 2.9 million customers, including customer names, encrypted credit or debit card numbers, expiration dates, and other information relating to customer orders. The government sector is also not an exclusion, in February 2014 data from 400,000 Austrian school tests are publicly appeared on Romanian servers [JN14]. With information about tests result during 2011 and 2012 years, in which nearly every second school were involved over these years.

These trends only grows with time. In 2013, 1143 leaks of confidential information were recorded and reported in the media by InfoWatch Analytical Center [Cen14a]. This figure is 22% higher than the number of leaks registered in the past year. All these problems are not only caused by human factor, but also the lack of appropriate countermeasures against such attacks.

1.1 Motivation

Databases and desktop have the highest rate of breaches among all business assets affected within insider misuse, according to the 2014 Verizon Data Breach Report [Cen14b]. The reason why databases are targeted is obvious. Databases are at the heart of any organization. They store customer, financial records and other confidential business data. Gaining access to sensitive information attacker or malicious insider can quickly extract financial values, inflict damage or have an effect upon daily business operations. The affected organization not only incurs a monetary and reputation damage, but also financial penalties from government due to regulatory violations. In very rare cases such violation is punishable by imprisonment [Bül10]. For example, Data Protection Law in Belgian provides criminal sanction up to two year and in Cyprus up to five years of imprisonment.

In the last decades, working environment of users is dramatically changed. From the beginning of 90th, where regular user had a fixed workplace and used specially configured hardware and software to our days, where user has flexible, mobile environment and can gain access to critical organization information from everywhere. This changes bring not only positives effects, but also bring downsides that more opportunities are created for insider attacks.

From the above information and security reports, we can see that while the insider threat has always existed in organizations, it has progressively become a serious issue of our time that must be better managed. However mitigating it is difficult process as insider threats are determined by a combination of technical, behavioral and organizational problems. Indeed, suspecting an outsider threat, security officer can at once isolate the system from the attacker, blocking the assumed attacker from accessing the database. However, such strategy does not work well in case of threats inside of the organization. Since blocking an insider prevents her to do the job the insider is responsible to take. Moreover, insiders, with their superior knowledge of the organizations inner control mechanism and security access systems, can freely navigate through the databases to retrieve the data they want and hide their intentional action.

In other words, mechanisms for protecting data from outside attacks are not always succeed in protecting data from authorized users who may misuse their privileges to steal or copy data from the DBMS. Therefore finding countermeasures that successfully protect sensitive data against insiders has become a key demand due to the amount of loss that can be caused by those malicious insiders.

One of such good countermeasures is implementation of defense in depth approach on DBMS (see on the Figure 1) with addition defense layer specially created and configured to detect and prevent possible data leakage.

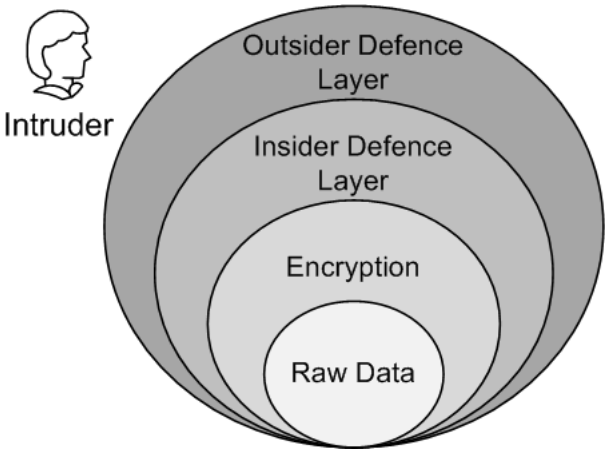

Figure 1: Defense in depth approach on DBMS

Certainly, controlling data leakage of DBMS works much better when it is done as close to the source of data as possible. In our work, we want to present a solution of aforementioned problem and create simplified DBMS platform with prototype of data leakage mitigation defense layer that demonstrates how using it the organization can prevent and detect insider threat.

1.2 Objectives and Sub-tasks

The objective of this work is to look deeply on the problem of insider threat, especially at the data leakage, explain common problems of this field, reasons why such crime exists and find a way to prevent those crime. For that reason, we perform an analysis of existing approaches and compare it with MVAL approach of Barthel and Schallehn [BS13b, BS13a]. Another objective is to develop a framework that can be used as a demonstrator platform for MVAL approach.

To do so, the following **sub-tasks** are deduced:

Task 1. Overview, comparison and evaluation data leakage countermeasures.

Under this task, the general introduction of insider threat has to be given as well as the research of existing mitigation approaches has to be provided and compared with MVAL approach. Also, missed points of MVAL approach have to be explored.

Task 2. Choose running environment for MVAL demonstrator platform.

Future platform must not have weak points of traditional native/desktop application and weak points of pure web application. Detailed description of requirements has to be presented in next chapters.

Task 3. Develop core DBMS functionality with support of SQL query language.

As the data leakage mitigation methods focused on prevention of sensitive data extraction from DB and not on modification or deletion. The core functionality must cover only various SQL extraction statements. The detailed list of functional requirement has to be given later.

Task 4. Integrate MVAL approach into developing DBMS engine.

The data leakage preserving by MVAL approach has to be implemented and integrated with DBMS engine of demonstrator.

Task 5. Design and implementation of usable Graphical Interface.

Under this task, the efficiency and effectiveness elements of graphical interface has to be examined and implemented into the application to improve overall usability for a user.

Task 6. Present evaluation of the implemented platform.

The final application need to be evaluated, based on requirements which must be defined during in the previous tasks.

After the objectives and sub-task are defined, the further structure of the thesis was developed and presented below.

1.3 Structure

The rest of the thesis is organized as follows.
In **chapter 2**, we give necessary terminology and fundamental basics about database design and architecture. Next, we introduce the traditional measures for database protection and provide technological overview of used information tools. After, in **chapter 3**, in the beginning we describe relatively new problem of insider threat in databases. Afterwards, the brief overview of countermeasures is presented and compared with MVAL approach of Barthel and Schallehn. Then, deep description of MVAL approach is given as well as step for improvements. In **chapter 4**, we consider different types of application architecture and choose one after analysis. Afterwards, we define requirement for the graphical user interface, database engine together with MVAL approach. By the end of this chapter, we give the schema of database. Later, in **chapter 5**, we describe the implementation of the demonstrator for data leakage preserving by MVAL approach. In the last **chapter 6**, the final conclusion about the work is given, as well as the suggestions for the future improvements of demonstrator.

2. Fundamental basics

In this chapter, we present the required knowledge, terms and their meanings for understanding the information of following chapters. From the chapter introduction, reader must understand that our work connected with DBMS security field. For this reason, we start briefly describing the necessary information about architectures, design and implementation of a database management system in the Section 2.2 and Section 2.3, and why they had a major influence on the growing use of computer system. Then in the Section 2.4, we explain well known techniques for securing databases against a variety of threats. As during development we used information technical tools, such as programming languages and external libraries, we briefly explain their roles in the Section 2.5.

We do not pay attention to every detail as the readers must have some background from information technology field. Nevertheless, if any questions appears beyond of what we explain, we refer readers directly to well known literature of database field [EN10, CB01, CB05, RGG03] and programming languages [Fla06, Mog10].

2.1 Basic Concepts of Database System

Databases and database systems are vital components of everyday life: most of us encounter several activities every day that involve some interaction with a database, without noticing them. For example, if we go to the bank to withdraw cash, if we access online library catalog to search for a book/magazine, or if we make purchase in e-commerce shop, the chance that our activities involves indirect interaction with database is very high.

The ”word” database, database management system are so commonly used, that from the beginning, we must defining what we mean by these terms:

Database

A Database (DB) is a logically coherent collection of related data that represents some aspect of the real world. It is designed, built, and fill up with information for a specific purpose. Usually DB consists of operational data and metadata, which is also divided into structural metadata ("data about the containers of data") and descriptive metadata ("data about the data content"). DB is not limited in size or complexity. From one side it can contains only few hundred records, names and addresses each with a simple structure. And from opposite side, the computerized catalog with billion entries for every citizen of the country organized under different categories [EN10].

Database management system

A DBMS is a software system that facilitates the processes of defining, constructing, manipulating, sharing data from a database in efficient way among various users and applications, protecting and maintaining it over a long period of time. It contains complete description of the database structure and constraints. There are many different types of DBMSs, ranging from small systems that run on personal computers to huge systems that dispersed over a network of interconnected computers [EN10]. Some of well known examples are MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, SAP HANA and IBM DB2.

The DBMS brings a lot of advantages [Cod82] and offer capabilities that provide environment for convenient working with data:

Controlling Redundancy - using data normalization approach, there is no need to duplicate data for every group of users, which cause a more efficient storage space consumption.

Storage Structures and Search Techniques - DBMS provides specialized data structures and search techniques to increase search for the desired records on the disk utilizing different indexes, buffering and caching techniques.

Integrity Constraints - DBMS provides capabilities for describing and enforcing integrity constraints on the data. A good example is uniqueness, referential integrity.

Actions Using Rules - in form of stored procedures. They are a part of the general database definition and invoked appropriately when certain conditions are met.

Security Access Restriction - as well as all data stored in one manageable place, it is easier to implement security and authorization subsystem to prevent unauthorized access.

Backup and Recovery - has facilities for recovering from hardware or software failures.

Concurrency control - ensure that concurrent or parallel transactions operate the same way as if they run in sequential order.

Multiple User Interfaces - query languages and programming language interfaces for application developers and menu driven interfaces for normal users.

Up to date information availability - as soon as the user updated some information inside DBMS, all other users can see the result of this update.

Transaction support - ensure that each or none operations inside the one transaction are reflected on the related databases. This ability provides to keep database consistent and provide ability to recover in case of system failure.

Database system

Database system (DBS) system is a general term and we use it to refer to the combination of a database with underlying data model, a database management system and application programs for interaction with DBMS. Figure 2 illustrates DBS and parts from which it consists.

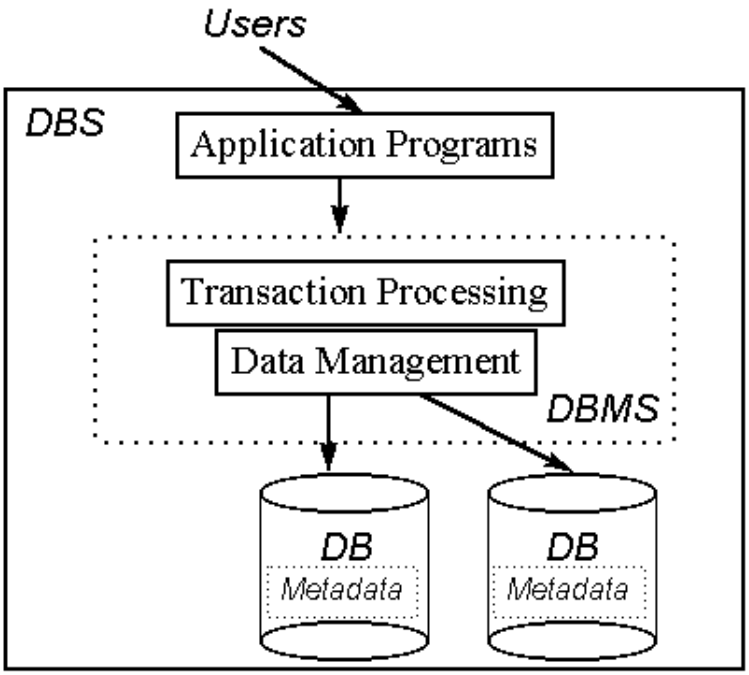

Figure 2: Structure of the database system [Hol13]

2.2 DBMS Architecture

After defining basic terms in DBS, we still need to explain details of fundamental architectural, theoretical and technical properties of DBMS, due to knowing them helps better understand developing demonstrator.
In this section we describe the typical architecture of a DBMS from different angles:

Application architecture - focuses on application uses.

Functional architecture - identifies the main components.

Logical architecture - describes different levels of data abstractions.

2.2.1 Application Architecture

The architecture of current DBMS evolutionized from monolithic systems, where all components were tightly integrated and run on one high powerful mainframe, to modular in design high distribute architecture. Where every module can be changed without affecting other modules.

Centralized DBMS architecture

In centralized database architecture, user data, application programs and DBMS is located in a one computer system. In early ages of development it was the most used type of architecture, but with increasing speed and bandwidth of computer networks appeared new type of architectures a client/server DBMS. But even nowadays, we can see DBMS like SQLite, which still follow the centralized principles.

Client/Server architectures

The client/server architecture was developed to deal with problem in computing environments in which a large number of personal computers (clients), servers and other computer equipment are connected via a network. The main idea is to specialize every component of the system and tune it to make their task as effective as it is only possible.

Figure 3: Three-Tier Client–Server Architecture

Every client of DBMS contains appropriate programming interface to utilize their service and to make processing this data using own resources. One of variants of such architecture is presented on Figure 3.
The presentation layer responsible for displaying information to the user and allows enter of data. The business logic layer handles intermediate rules and constraints before data is passed to the user or to the DBMS and the database service layer in charge for all data management services.

Parallel database system

The main goal of **parallel database system** to improve general performance through parallelization of input/output and processing operations. It is possible by using simultaneously multiple processing units and disk storages. Furthermore, they provide the higher level of availability, as in case of fail of some components the DBMS continue working. Such systems can be divided in following groups: shared memory architecture, shared disk architecture, shared nothing architecture.

Distributed database system

The data in **distributed database system** is physically stored across several geographically separated sites. Usually each site is managed by a DBMS that is able to run independently from the other sites. In comparison to parallel databases, the main factor is to increased availability, in addition to performance issues.

2.2.2 Functional Architecture

Functionally, every DBMS consists of following main components shown in Figure 4.

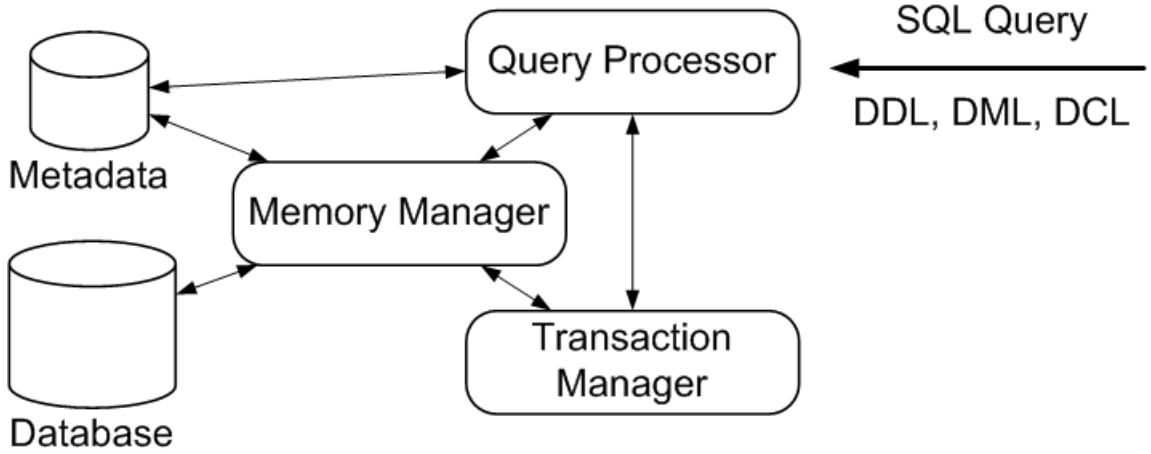

Figure 4: Functional architecture of DBMS

Query processor transforms and optimizes a user query into instructions the DBMS can process them efficiently. It takes into account the metadata of DB.

Memory manager retrieves and store data from the database that satisfied queries compiled by the query processor. It keeps metadata up date with changes to DB and manages the structures that contain data, according to the Data Definition Language (DDL), Data control language (DCL) directives.

At the end, **transaction manager** ensures that the execution of possibly many transactions on the DBMS satisfied the atomicity, consistency, isolation, and durability properties. Moreover it also provides facilities to recovery the system and media in case of failures.

2.2.3 Logical Architecture

Logical architecture is also known as the ANSI - SPARC (American National Standards Institute, Standards Planning And Requirements Committee) architecture, was invented at the beginning of 70s. Modern DBMS are based on this architecture, but it is never became a formal standard. The goal of such schema is to separate user application from the physical realization of DB, it is illustrated on the Figure 5. Most modern DBMSs do not explicitly separate the three level architecture, but support it to some extent, for example combining internal with conceptual level.

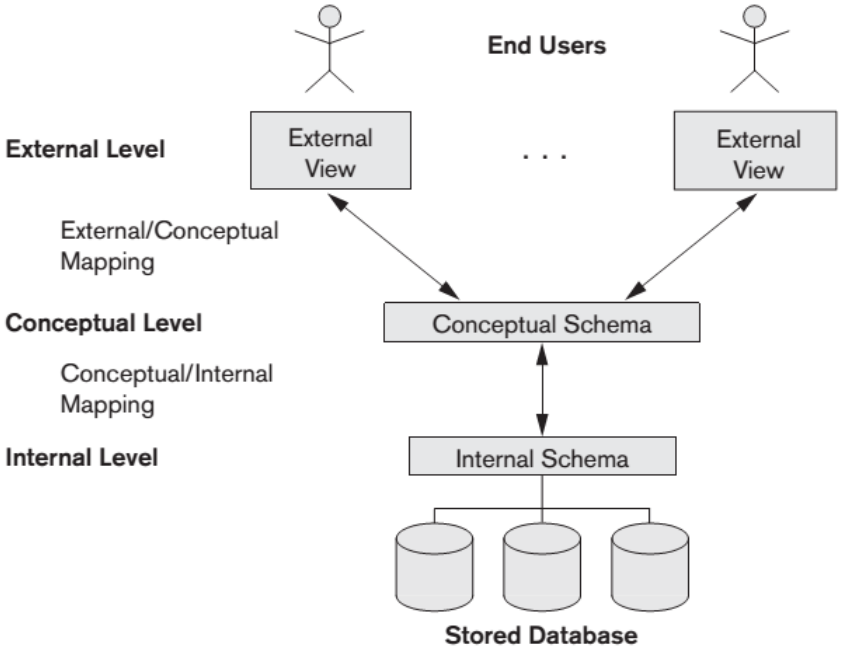

Figure 5: ANSI/SPARC or three schema a architecture[EN10]

It distinguishes three layers of data abstraction:

- Internal level - describes physical stricture of DB. Using physical data model to define complete details of data storage and access paths to the DB on the computer system.

- Conceptual level - describes data, which is stored within the DB and how the data is correlated with each other. Conceptual schema hides the details of physical DB realization. Furthermore, it concentrates on describing essential data elements that need to be stored and the relationships between them (entities, data types, relationships, user operation, constraints). Traditionally for describing conceptual schema used one of the data models (see details in Section 2.3) and for it design is based on conceptual schema design a high level data model. Most of the time on this level work Database administrator (DBA).

- External level - also is called as users views of the DB. It describes a part of the database that is relevant to a particular user and hides the reset of irrelevant data or data the user is not authorized to access. In the same way as with conceptual schema, external schema is implemented using an external schema design in a high-level data model (see in Section 2.3).

Due to the presence of three different levels, a DBMS must transform user request on the external level into a request on the conceptional level and then into the request on the internal level for further physical processing. The same path, but vice versa goes the request result to the user. This process of requests and results transformation between levels is called **mapping**.
The separating of the internal level from the external level eliminates necessity for the user to know how the data is physically stored in the database and allows DBA to change the database internal structures without altering desired individual users view. This architecture allows to realize two basic principles of data independence:

- Logical data independence - change only the conceptual level, without any change in the external or in user view.

- Physical Data Independence - change the internally without need to change the conceptual level, externally or user view.

To conclude the above explained, three-level architecture makes possible to achieve data independence both physical and logical, but create small computational overhead during mapping user query between levels.

2.3 Database Models and Query Languages

A **data model** is a collection of high-level data description constructs that hide any specific implementations of low-level storage details or protocols used to transport the data [EN10]. It is a fundamental entity to introduce abstraction in DBMS. The primary motivation for such abstraction is to formalize the description of a domain problem without restricting how that description will be mapped to an actual implementation in software. In addition to that a DBMS constrains a user to store data in terms of a used data model.

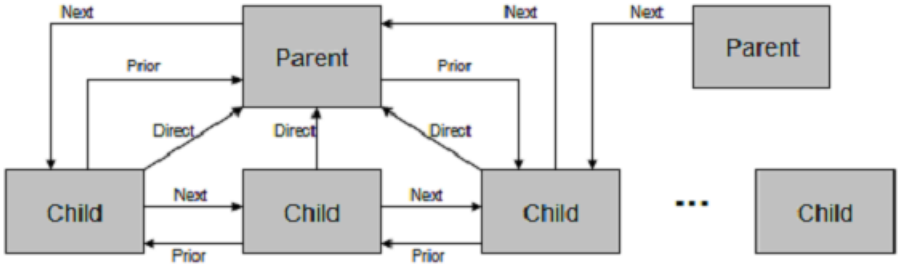

Figure 6: Network data model [SPC+10]

Depending on the implementation, DBMS can support multiple data models simultaneously. The most popular one is **relation data mode**, but it is not the only one. We deeply describe in the next sub chapter.
First invented models were **network** and **hierarchical data model**. Illustration of them are presented on the Figure 6 and Figure 7 accordingly. DBMSs which implemented them belong to the so called legacy database systems, they are still used by governments and business, due to old applications tightly bind with them, but the number of new installation is very low, because many of its users have converted to relational systems [EN10].

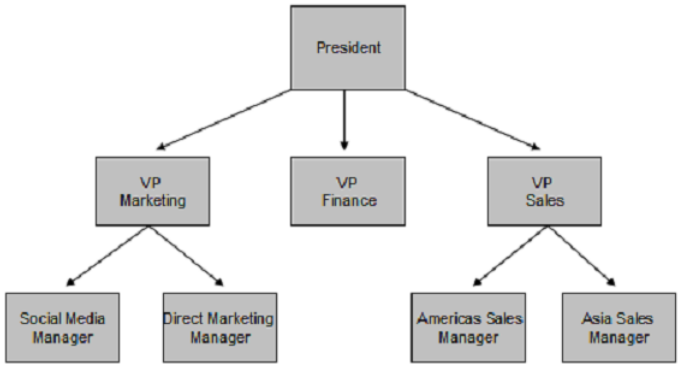

Figure 7: Hierarchical data model [SPC+10]

After relational model in the mid 70s, there was proposed the **entity-relationship data model** [Che76]. Instead of being used as a data model on its own, the entity-relationship model has found success as a tool to design relational DBs.
To the family of highest level data models includes **object** and **object relational model**. Every entity in it treated as an object and a relationship as an inheritance. They are still not so widely used in the industry and found their accepted mostly in research institutes and universities.
In the last decade, new models are appeared such as semi-structured, semantic and object oriented data models, but they are beyond the bounds of our work.

2.3.1 Relational Model

Relational data model was first introduced by Codd in 1970 during his work in IBM [Cod70], illustration of it on the Figure 8. It attracts immediate attention due to simplicity and exceptional mathematical foundation.
In the relational model [RGG03], all data are stored in relations or table. Each relation consists of rows and columns. In addition to that each relation contains a header and a body. Header is a list of columns in the relation and body is set of data that actually occupies the relation, organized into rows or tuples. The second major characteristic of the relational model is a key usage. Keys are specially defined columns within a relation to order data with other relations. Primary key is the most important one, which is used to uniquely identify each row in relation. Foreign key is a column or set of columns in one relation that uniquely identifies a row of another relation. Besides defining how the data are to be structured, the relational model has a set of rules to enforce data integrity, known as integrity constraints.
A DBMS that implements above described model is called a Relational Database Management System (RDBMS). In the same way, when a DB is formally described and organized according to the relational rules also called Relational database (RDB).

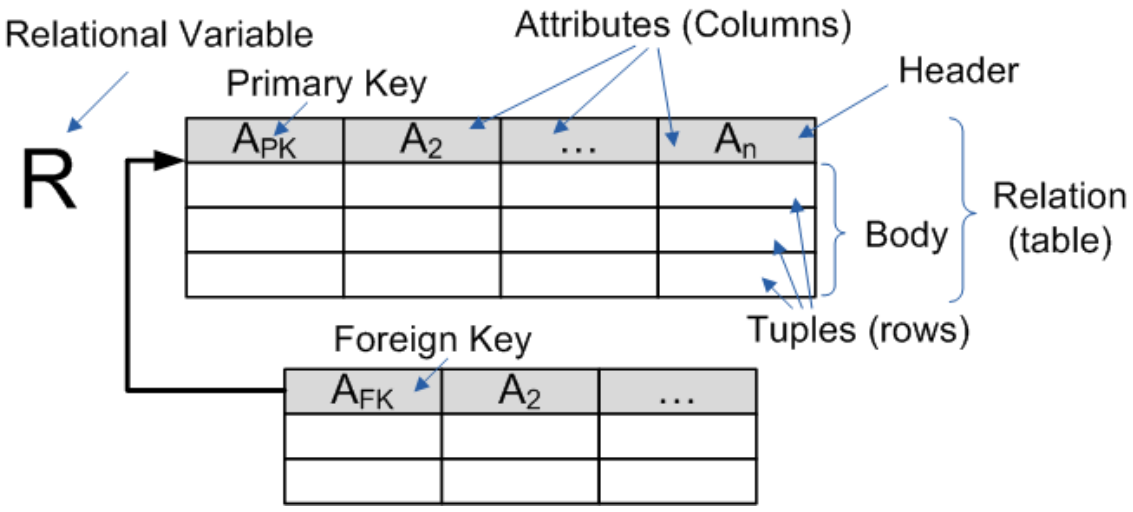

Figure 8: Relational data model with referential integrity constraints

Also important to notice, that author of relational model Edgar Codd, after extensive research came up with twelve rules [Cod85], which according to him, a database must follow in order to be a true RDBMS. These rules were result that in early day of his invention, many products were released as RDBMS when they were not [EN10].

2.3.2 Relational Algebra

One of the two formal query languages[1] associated with the relational model is **relational algebra**. Queries in relational algebra are composed using a collection of

[1]Another formal query language is relational calculus

operators. A fundamental property of every operator is that they accepts as arguments a relation instances and returns also a relation instance as the result. This characteristic makes it possible to combine operators to form a complex queries, they are known as **relational algebra expression**.
There are two group of operations:

Unary relation algebra operator - applied to a single expression, for example Projection, Selection, Renaming.

Binary relation algebra operator - applied to two expressions, for example Product, Union, Intersection, Division. Major of them were copied from the mathematical set theory.

The procedural nature of the relational algebra allows us to think about relational algebra expression as a plan based on which we compute the desired answer. For this reason algebra expression are used to evaluate and optimize the query plan.

2.3.3 Structured Query Language

One of the major reasons for the commercial success of relational databases was appearance of SQL language. It was originally developed at IBM as part of System R project in 1970s. Almost immediately other vendors introduced DBMS products based on SQL and it is became a de facto standard. SQL was adopted as a standard language in the mid 80s by the American National Standards Institute (ANSI) and then by the International Standards Organization (ISO). It continued evolving in response of changing needs in the database community. SQL allows users to create DB, tables (relation structures), perform modification and retrieval information through queries. SQL query is a group of commands that is passed to the DBMS, where user specifies which data must be gathered from which tables and how it should be arranged.
SQL statements are traditionally divided into following categories:

- Data Definition Language (DDL) - this part of SQL supports the creation, deletion and modification of definitions for tables and their views. Moreover using it user defines internal integrity constrains.
- Data Manipulation Language (DML) - allows users to retrieve, insert, delete, and modify rows.
- Data control language (DCL) - provides mechanisms for specifying access rights to data objects such as tables and views.
- Transaction Control Language (TCL) - provides commands to allow a user to explicitly control aspects of how a transaction is executing.
- Session Control Statements - allows to manage properties of a particular session of a user.

Further we present SQL aspects that must help readers to understand functionality boarder of the developing framework.

NULL value

SQL provides a special value called NULL to use in such situations when column values can be unknown or inapplicable. The principle is similar to object oriented programming languages, when you do not initialize object during creation and its default value is NULL.

Three valued logic

Due to null values that we have just introduced, definition of logical operators must be done using a Three Valued Logic (3VL) in which expressions evaluate to the true, false or unknown. 3VL builds upon Boolean Logic. The truth tables for 3VL AND, OR and NOT predicates are shown below on Figure 9. 3VL has impact on all boolean,

p	q	p OR q	p AND q	p = q
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT p
True	False
False	True
Unknown	Unknown

Figure 9: Truth table for three valued logic, modified [Sql10]

arithmetical and duplicate elimination operations. A good example is the WHERE clause condition. It must eliminates rows for which the qualification does not evaluate to the ”True”. For this reason with the presence of null values any row that evaluates to false or to unknown is eliminated from the result relation.

Nested queries

One of the most powerful features of SQL is nested queries, presented on Listing 1. Using them we can embedded query into another query, where the embedded query is called a sub query respectively. Sometimes when we write a query, we need to express a condition that refers to a table that must itself be computed. The query used to compute this auxiliary table is a sub query and typically appears in the WHERE clause of the main query.

```
1 SELECT attr1, attr2, attr3
2 FROM talbe1
3 WHERE talbe1.attr1 IN (SELECT attr5
4                        FROM table2
5                        WHERE table2.attr1 = 100 )
```

Listing 1: Example of nested query

2.3.4 Logical Query Optimization

There are many ways how even simple query can be executed, each of which is superior in certain situations. The aim of DBMS to consider as many alternatives as possible to choose the one with the least estimated cost. Queries with different operations have many evaluation options and process of searching a good plan is a serious challenge.

Applying transformation rules, the DBMS optimizer transforms one relational algebra expression into another that is equivalent to original. There are several dozen rules [CB05], for an example we provide some of them:

- **Conjunctive selection** - selection operations can cascade into individual selection operations and vice versa.

$$\sigma_{c1 \wedge c2 \wedge c3}(R) = \sigma_{c1}(\sigma_{c2}(\sigma_{c3}(R))) \tag{2.1}$$

- **Conjunctive selection, join, product** - changing the order of the operands does not change the result.

$$\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R)); \quad R \bowtie S = S \bowtie R; \quad R \times S = S \times R \tag{2.2}$$

- **Commutativity of Selection and Projection** - only in case when the condition predicate involves only the attributes in the projection list.

$$\Pi_{A1,\ldots,An}(\sigma_c(R)) = \sigma_c(\Pi_{A1,\ldots,An}(R)) \qquad where \;\; c \in \{A1, A2, \ldots, An\} \tag{2.3}$$

Instead of exhaustively enumerating possible combination of rules and choosing the cheapest among them directly, the search space was limited in advance by means of logical rewriting. They are so called heuristic rewrite rules, applying them we lead to equivalent expression but with known a better efficient executing cost. Further we want to present some of heuristics rules that could be applied during query processing:

- Perform **Selection** operations **as early as possible**. Performing selection we reduce the cardinality of the relation as well as reduce the subsequent processing of that relation.

- Perform **Projection** operations **as early as possible**. The same like with selection, it reduces the cardinality of the relation and reduces the subsequent processing of that relation.
- **Most restrictive join** operations are executed first, general rule of thumb is to perform most reduction of data before performing other binary operations.

The logical optimization is only the starting point for a next phases : cost-based or systematic optimization [Gra93, Cha98].

2.4 Database Security

Due to increased development of information technology and their integration with work flow within organization, it caused an accumulation a huge volume of data about every aspects of their activities. Using it organizations make critical decisions for their future existence, which leads that data becomes an extremely valuable asset. Everything, that has an affect on security of data become a potentially source of threat. As a result, we must pay a close attention to data security and provide complex measures to prevent possible data misuse or its destruction. When we talk about database security, the protection should not only be applied to data inside a DB, but also to all part of the system that effects data. It also includes hardware, software and staff who are in contact with it.

In this section we describe scope of DB security and identify range of typical threats and their consequences on DBS and organization, as well as known detection and prevention measures against them.

2.4.1 Threats

According to Connolly [CB05] in general **threat** is "*Any situation or event, whether intentional or accidental that may adversely affect a system and consequently the organization.*". Three key characteristics of data or services that must be protected by information security, are illustrated on Figure 10.
Correspondingly, database threats are the result in the partial loss or degradation of some or all fundamental security characteristics: Confidentiality, Integrity and Availability (CIA), illustrated on the Figure 10. Achieving protection against them is not possible without strict and clean security policies that determined which users can access to which data in database and using which operations. Below, we present what can happen in the case of loss one of the CIA qualities [CB05].

Loss of integrity - refers to the requirement that data must be protected from improper modification. It means, that data are still present in a DB, but have partly becomes corrupted or invalid by the unauthorized changes to it either intentional or accidental.

Loss of availability - refers to cases when data or system is not available to user or a program while they have a legitimate right access them. What can lead to seriously financial losses for organization and in some cases to data corruption.

Loss of confidentiality - happens when data stored in DB can be viewed by individuals who must not have access to it. Unauthorized or unintentional disclosure could have negative impact on reputation of organization or legal action against the organization.

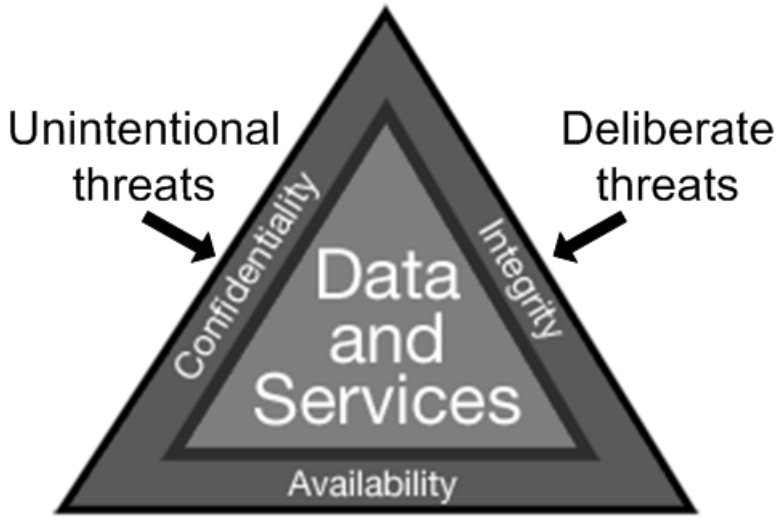

Figure 10: Information security triangle (CIA)[NC13]

These qualities represent areas in which we must take close look to reduce risk, that is the possibility of incurring direct damage for the organization. For the protection again aforementioned threats, DBMS must implement following countermeasures [EN10]:

Access control - restrict access to the information inside of DBMS. More details in the Section 2.4.5.

Inference control - aims to eliminate indirect disclosure of information, controlling dependencies among data DBS. More details in the Section 2.4.6.

Flow control - regulates how data flows between available objects of DB.

Encryption (Section 2.4.7) - cryptography techniques for protecting data outside of DBMS. More details in Section 2.4.7.

Before going deeply in details of possible countermeasures, we need to understand, that there is no need to protect data when nobody has interest in it. Further in the work, we describe, what kind of data is primary target for the attacker.

2.4.2 Data Sensitivity

In everyday life, we primary face public available data like articles, images in the Internet. Besides, there is other type of data, like highly important to government or military exists and to which a limited number of people is allowed to have access. For this type of highly importance data owner must assigned a measure of the importance or sensitivity, for the purpose of denoting its need for protection. The following factors are caused data to be classified as sensitive :

- **Inherently sensitive** - the data value by itself may be confidential, that make it automatically sensitive, like a bio-metric data of a person.
- **Declared sensitive** - the owner of data decided, that it is sensitive, like result of research, intellectual property.
- **Sensitive source** - the data was taken from confidential source and by the nature must be also sensitive, a good example when buyer pays at e-commerce shop using credit card information.
- **Sensitive in relation to other data** - some data can become sensitive in relation and presence of other data, but without it does not carry any importance. For example company presents a new product on the market, but if we do not know exact time and date, then such information become less sensitive.
- **Sensitive attribute** - particular record might be declared sensitive, such as credit history in a bank DB.

Next, we look at typical countermeasures for data protection in organization.

2.4.3 Authentication

The first step in providing security is controlling who is allowed to access the DBMS. This process is called **authentication** by which we verify, that someone is who they claim they are. The most common and cheapest way is to ask for user's name and password, but it can also include other methods that uniquely identify a person such as checking smart card or biometric information. The result of the authentication process is usually a "YES" or "NO", either the user identification is verified or not.

2.4.4 Authorization

The second step in providing security is controlling to which resources the authenticated user is allowed to access in the DB. In addition to that, authorization verifies either user has enough credential to perform a specific action (create, read, update or delete) over requested resources. The level of authorization for each user is determined by additional properties associated with the account of the user. For an example, due to administrative policy inside a company, it is not allowed to work on weekends, when

user come on Saturday morning, she will be authenticate but not authorized to access the system.

Authentication and authorization process is vital part of every access control system which we deeply describe below.

2.4.5 Access Control Countermeasures

An access control model is a framework that include together authentication, authorization and additionally to that defines how user (subject) access data (objects). Using access control technologies and security mechanisms of DBMS to enforce the rules and objectives of the model [Gen12]. There are four main types of access models:

- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)
- Role Based Access Control (RBAC)

Further we look deeply at each of these and explain what they entail.

Discretionary Access Control

DAC system is based on a fact, that each resource object has an Access Control List (ACL) associated with it, traditionally it associated with relation or views. Illustration of the DAC is presented on the Figure 11.

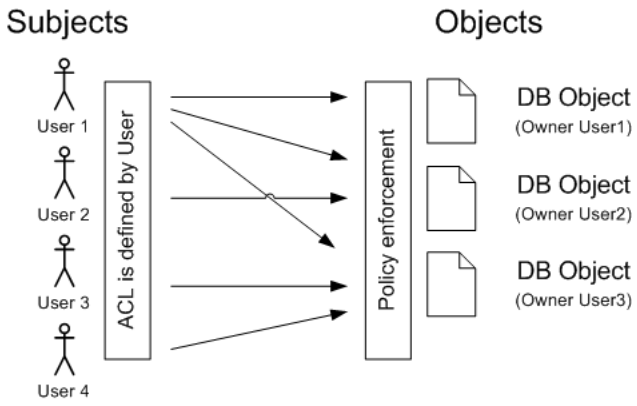

Figure 11: Discretionary Access Control

Another important fact, that under DAC a user who owns the object is responsible for setting permission on this object. Support for DAC was included in the standard of since version SQL-92 through the GRANT and REVOKE commands. Therefore DAC techniques of granting and revoking privileges have been the main security mechanism

for DBMS.

While being effective and easily understandable this approach has certain weak points. For an example, DAC is vulnerable to malicious attacks, like Trojan Horses embedded in normal application. When victim executes such software and in addition to normal behavior it grand additional permission to object with sensitive data to the attacker. Another drawback is that there is no formal definition for unauthorized flow of information in the system that caused MAC model to appear for overcome this problem.

Mandatory Access Control

MAC is one of the strictest of all access control measures and primarily used by government or military for protecting their data assets. In opposite to DAC where object assigned to subject (user or program), here in MAC objects (views, relations, tuples and attributes) are assigned to **security class** and each subject is bind with **clearance** for a security class. In addition to that MAC is based on system defined policies that cannot be changed by individual subject, one of them is known as Bell–LaPadula model [Bel05] and has following restriction properties:

- **Simple Security Property** - a subject at a specific security level must not read an object at a higher security level (**no read-up**).
- **⋆-property**[2] - a subject at a specific security level must not write to any object at a lower security level (**no write-down**).

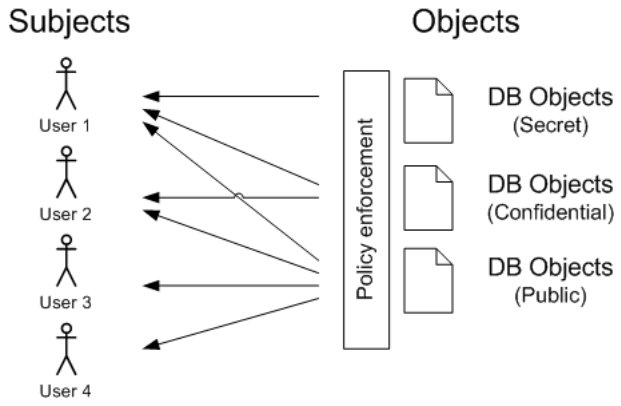

Figure 12: Mandatory Access Control

For explanation (see Figure 12) of these rules and policies, let us define security class: top secret (TS), secret (S), confidential (C) and public (P). Using first rule, object with security level (TS) must not be read by any of subjects which security clearance level is

[2]Read ”star”-property

below (S, C and P). The second rule is less intuitive, it prohibits subject of high level (S) from writing to an object at lower security classification (C and P), then a subject itself. These rules provide multilevel security and prohibit unauthorized information flow in the DBS.

Nevertheless, MAC has also drawbacks. It requires a considerable amount of time for planning a rigid classification of subjects and objects into security levels before it can be effectively implemented, that makes it applicable only to a small number of environments. Consequently, there is no official SQL support for MAC model. In many practical situations, DAC is preferable due to its trade off between security and application.

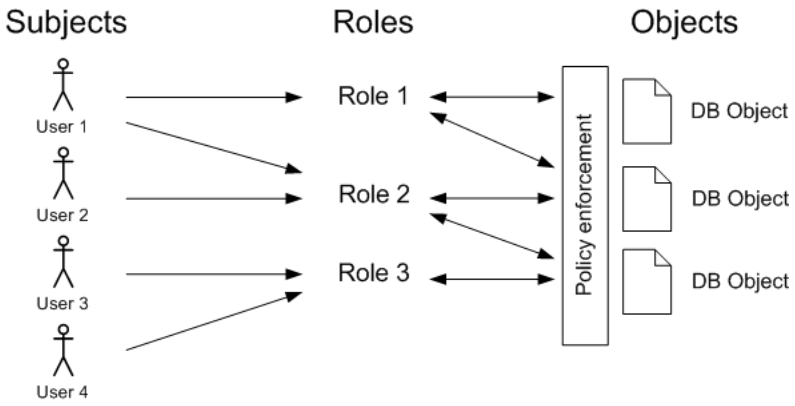

Figure 13: Role Based Access Control

Role Based Access Control

RBAC model was standardized by National Institute of Standards and Technology (NIST) in 2004 [INC04] and provides a proven technology for managing and enforcing security in large scale systems [FK09, SFK00]. RBAC is also known as non discretionary access control, presented on the Figure 13. Privileges and permissions are associated with specific organizational role and then user is assigned to this appropriate role. Another important aspect of RBAC is support of hierarchy and rights inheritance, when rights flow to the bottom in the hierarchical structure, users lower in the structure gain the accesses granted them from the above. A typical example is the relation between an student and a manager of in the project. The manager needs to access the data that the engineer has the access too and in the same time to the data that is of more administrative nature. It means, that rights associated with the manager role encapsulates the rights that are associated with the engineer role. Another advantage of using RBAC is that it becomes very straightforward to add new user or modify access of user, as the only action required is to change the taken roles of the user. RBAC can be build upon traditional DAC and MAC. In this case privileges already held by the user are

merged with privileges that are included in the specific role.

RBAC, MAC and DAC are most widely used access control models, but several other exits such as Biba integrity model [Bib77], chinese wall [BN89] and label-based [PN09] security models.

2.4.6 Inference Control Countermeasures

Access control models protect sensitive data against unauthorized disclosure using direct access rule and policies, nevertheless it fails to prevent in case of indirect access. Indirect sensitive data disclosure is possible, when access to it is possible via non-sensitive data and meta data (see Figure 14).

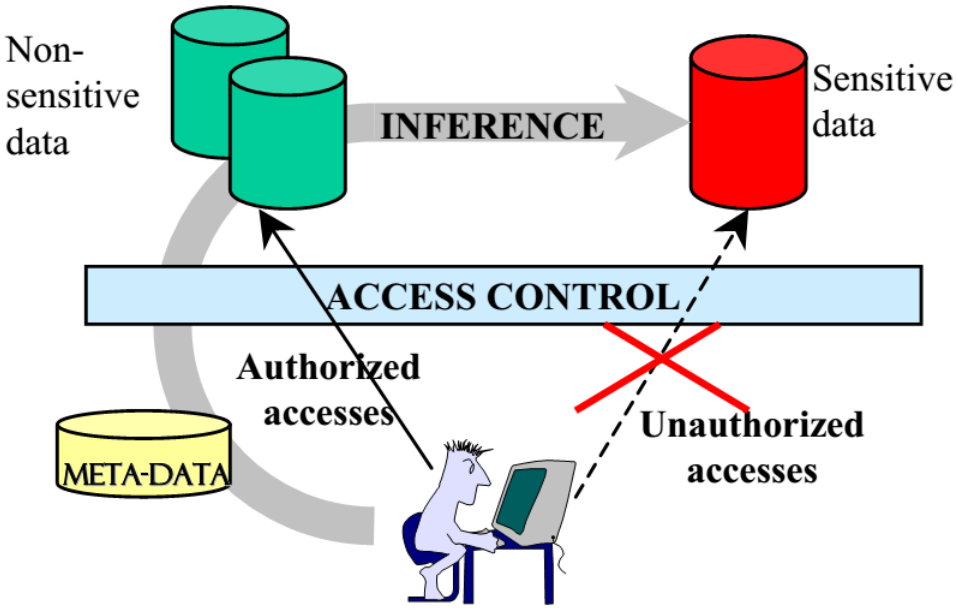

Figure 14: Access to sensitive information via inference channels [FJ02]

The inference controls countermeasures mainly applicable to statistical and multilevel secure databases.

Statistical databases

Statistical DB is first type of DB where inferential violation were investigated. The main security requirement is to provide access to statistics about groups of entities and at the same time protecting confidentiality of the individual entity. The problem of this domain, that user queering DB can obtain confidential individual information by correlating different statistics. The possible countermeasures include, prohibiting entire result from DB, when the number of tuples in the result relation is below pre-configured

thresholds. Another one is when result is combined with some noise to produce slight inaccurate result set that would be enough to prohibit accurate extraction of individual information [GTD98]. Another alternative is partitioning individual information into groups and only provide an answer based on this group, but not in subset of records inside the group [Lun89, LJ92].

Multilevel secure databases

Most of the inference channels in multilevel secure DB are created by combining meta data with non sensitive data in order to obtain information that has a higher security classification. Techniques for removing possible inference channels are base on utilize functional dependencies in the DB schema [Hin88, Bin92, Mar96] and data level [Den85, YL98].

2.4.7 Encryption

Encryption is used in the DBMS to protect data stored in the DB in cases when the aforementioned security measures are not enough. For example, when the intruder has access to the communication network of the user, she can steal critical data transmitted across it, such as user login and password and access database on behalf of authenticated user. Another example, when intruder had a direct physical access to backup of DBS, stealing them she would have complete copy of original DB.

Encryption is the process during which underlying data is encoded by a particular algorithm (a well known are Data Encryption Standard (DES), Advanced Encryption Standard (AES)), what makes it impossible for human or program to read data before being decrypted. If the access controls are bypassed or can not be used, we use an encryption to enhances security and privacy of data by prohibiting unauthorized access to it.

2.5 Technology Overview

In this section, we present overview of information technology that we used during design and implementation of the demonstrator application.

2.5.1 JavaScript

JavaScript language was developed by Netscape, at the beginning it had different names (Mocha, LiveScript) then after license agreement between Netscape and Sun it was finally renamed to JavaScript. The main idea at the start of JavaScript was to create language for web designers, people who may not have much programming skill [Fla06]. The language that allows them to add a little bit of animation or a little bit of smarts to their web pages and forms. JavaScript very quickly gained broad success as a client-side scripting language, that force another huge software company Microsoft to develop a

similar compatible dialect, they called it Jscript due to copyright issues. In June 1997 after Netscape submitted JavaScript specification to European Computer Manufacturers Association (ECMA) International for consideration as an industry standard, the first edition of it with small changes was released with the name ECMAScript (specification ECMA-262, ISO/IEC 16262).
ECMAScript is not a programming language, it is only a standard (latest version is 5.1), while JavaScript and JScript are languages that implemented this standard and both add additional features that are not part of the ECMA specification.
JavaScript popularity has been growing in the recent years and it has been claimed as the world's most popular programming language[Cro08]. Almost every computer device today (notebooks, tablets, smart phones) have at least one JavaScript engine installed, most often inside the browsers. There is a big competition between browser developers, they try to increase JavaScript engines performance as the most important features during product release, hence the performance of ECMAScript interpreter from the beginning was slow, but at the same time from it depends overall end user experience when working with web documents. Using SVG, audio, video, 3D and other similar multimedia elements in the web forced browser developers to provide JavaScript direct access to device hardware to gain higher performance, it is also called hardware-acceleration in web browsers [Net12].
Also important to notice, that JavaScript is not truly object-oriented, it has full support of polymorphism, however concept of inheritance and encapsulation is not directly implemented in the language, but it is possible to realize it using programming design patterns.

2.5.2 jQuery

jQuery is a cross-platform open-source[3] JavaScript library that was designed to simplify routine operation during client-side scripting of web page. It is most used JavaScript library, approximately 60% of websites in the world used it [Sur14b].
Some of most important features :

- Provides simplified DOM element selections, traversal and manipulation based on node element name, id or class criteria (details what is DOM in Section 2.5.4)
- Extended event handler
- Support of effects and animations
- Support of asynchronous JavaScript, Extensible Markup Language (XML) and JavaScript Object Notation (JSON) open standard format
- Broad support of plug-ins

After the main features of jQuery library is described, we explain the details of another library. Both of them, we use intensively during the development of demonstrator.

[3]Under MIT License

2.5.3 Relational Algebra Toolkit

RAT is an cross-platform open-source a web oriented software library written entirely in JavaScript. Its aim to help in explaining, teaching and learning the concepts of relational algebra in the RDBMS. They use own proprietary format for describing internal information. But it is based on the current XML standard for data exchange in the web.

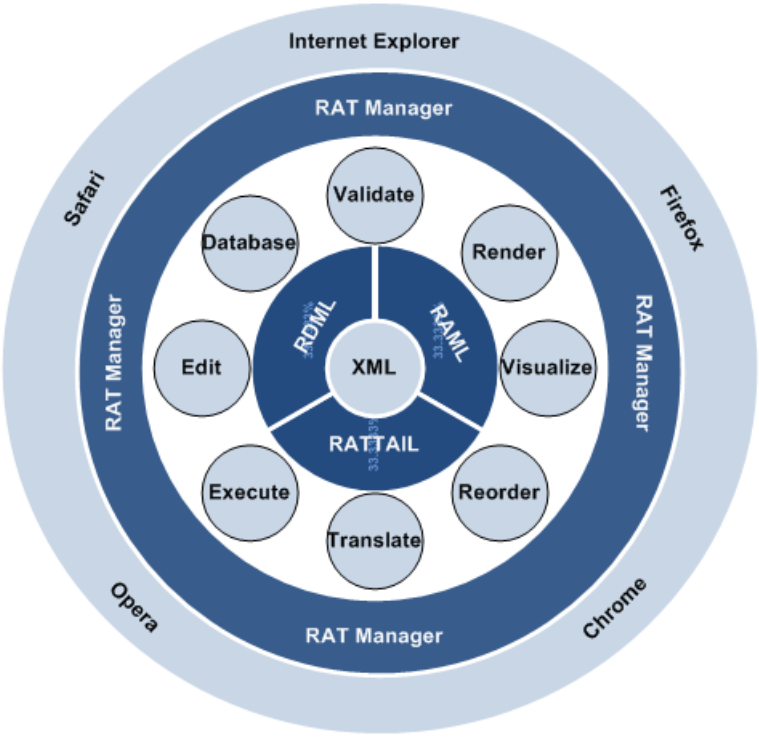

Figure 15: Layered architecture of the RAT [AC13]

RAT has a layered architecture (see Figure 15). At the core is the XML based languages that are used to represent the input of the user to the system:

- Relational Algebra Markup Language (RAML) - used for describing relational algebra expressions.
- Relational Database Markup Language (RDML) - used for describing entire relational databases as well as individual relation and their content.
- Relational Algebra Toolkit Automated Instruction Language (RATTAIL) - used for managing information flow inside of the RAT library.

The library is built from independent modules that are executed by RAT manager module in case the user call them in the RATTAIL script. RAT manager module is the coordinating process of the whole system and following services :

RAML Render - allows to present relational algebra expressions as part of the text content of a Web document. Relation algebra is encoded as RAML expressions and then converts by render module to a string of serialized Extensible HyperText Markup Language (XHTML) for further injection into web document.

RAML Visualize - allows present complicated relational algebra expressions in a way of expression tree. It use the same principle as render, but instead of XHTML output, it produces SVG HyperText Markup Language (HTML)5 element. SVG elements can be resized in depends on the free space on the web document without blurring or any loss of information.

RAML Reorder - manipulates RAML expression, permuting elements of it, until it produce the same result as original expression. The aim of it to provides different view on possible executing strategy for query.

RAML Translate - allows to convert RAML expression into an equivalent SQL query.

RAML Execute - allows invoke RAML encoded relational algebra query upon RDML encoded relational DB.

RDML Data Render - the functionality is similar to Render module, but instead of producing relational expression, it renders XHTML tables that visually represent the encoded resulting relation.

RAT Validate - check the syntax of RAML and RDML XML documents.

At this step the description of JavaScript and its extension libraries are finished. Next, we present general information about tools for creation and manipulation of web documents.

2.5.4 HyperText Markup Language 5

HTML5 is the latest evolution of the HTML standard that was not yet fully approved by World Wide Web Consortium (W3C) and exists as a ongoing draft [RB14]. In addition to the markup functions, HTML5 extends previous standard HTML 4.01 with a number of Application Programming Interfaces (API), which heavily interconnected with CSS 3 (Section 2.5.5) and JavaScript language (Section 2.5.1). As a result, part of new standard is implemented in JavaScript libraries integrated in the browser [Jär11] and some of these features are only accessible trough JavaScript API. That made scripting language an essential part in web application development.

HTML5 gives web applications the ability to look and have similar usability as traditional desktop applications rather than static text-image documents interconnected with links from which majority of old web site is built. Example of such applications

are online word processors (Google Docs, Microsoft Office 365), online e-mail application, social network sites. HTML5 defines new DOM API for server-side events, drag and drop, drawing images (SVG), video. Exposing theses new objects via DOM to JavaScript makes it easier to write desktop like applications, using tightly specification standards rather than proprietary documented functions.

Document Object Model (DOM)

Document Object Model (DOM) is language and platform independent software interface that provides a structured representation of HTML, XHTML and XML documents. It defines a way that the internal structure of documents can be accessed from programs in a way that they can change the document structure, style and content. DOM repre-

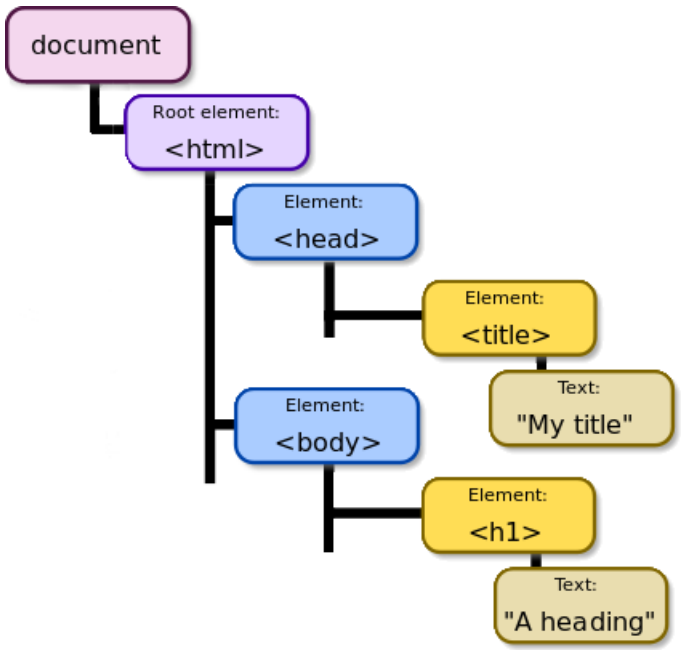

Figure 16: Typical DOM of a web page [Eri12]

sents a document as a structured group of interconnected nodes (Figure 16) and objects that have properties and methods and play important role in connecting underlying document structure with scripting language. The last release of DOM specification is Document Object Model level 3, published in April 2004.

Scalable Vector Graphics (SVG)

Scalable Vector Graphics or SVG is a language for describing two-dimensional graphics in XML. It supports grouping, transforming, compositioning and styling three types of graphics objects : vector graphic shapes (strength and curve lines), text and raster graphics. Besides that it allow to apply different effects such as clipping path, image filters, alpha masks.

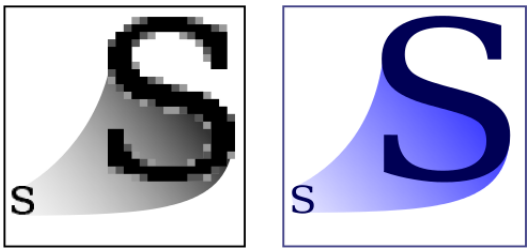

Figure 17: Differences between bit-mapped and SVG images [Yug06]

SVG provides resolution independent graphics for the web, print and on mobile devices in a compact format. Comparison of traditional raster graphic and SVG is presented on Figure 17. When SVG is used inside web documents, it is possible to style it using Cascading Style Sheets (CSS) and animate using scripting languages. It is royalty-free, vendor neutral open standard developed under the W3C, which widely supported by modern web browsers[Sta14].

2.5.5 Cascading Style Sheets

Cascading Style Sheets (CSS) is a language for describes the presentation of Web document, including panel layout, fonts, color, size of elements. Purpose of CSS is to allow web designer to reuse of design templates and adapt the presentation layer to diverse types of display sizes.
Three different levels of CSS standard exists, where each new level is built upon previous one adding new functionality and features. The latest revision of CSS specification is level 3. CSS is independent from HTML and can be used with any XML based markup language. Another important aspect of CSS is ability to separate presentation layer from content, this is a design philosophy and core methodology that applied in the context of different publishing technologies.

2.5.6 Extensible Markup Language

Extensible Markup Language (XML) is a markup language that defines a set of rules for describing electronic data in the readable form both to computer programs and humans. It is successor of another early popular markup language called **Standard Generalized Markup Language** or **SGML**. It is extensible because it does not have a fixed, predefined format like HTML. Instead, using XML we can design our own markup languages for documents.

At this chapter we gave necessary terminology and fundamental basics about database design and architecture. Besides, we introduced the traditional measures for database protection and provide technological overview of information tools, which we use to implement the demonstrator. In the next chapter, we deeply describe insider threat problem in DBMS, especially problem of data leakage. We present Barthel and Schallehn [BS13b, BS13a] data leakage preserving by MVAL approach to solve this problem. Afterwards, we present missed points of MVAL approach for further improvements.

3. Insider Threat Prevention Mechanisms

In our days, the DBMS are used in every organization, as it is the most convenient and efficient way how we can retain and share data electronically. The number of data in them is growing steadily every year [GR12]. As a result, the necessity for keeping data secured from unintended access is growing. Furthermore, this task becomes even more challenging in large distributed systems, where multiple organizations are involved with hundreds users in the same collaborative enterprise. The most widely used solution is implementation of access control mechanisms [PG06] (see details in Section 2.4.5). But they make decisions either to show data to user or not, on pre-configurable set of rules. Once a user is authenticated by the system, authorization set of privileges is assigned to a user until log out. Any changes in user work behavior, location, from where user connects, computer environment is not considered.

For example, if the person is going to leave a company, what will stop her from copy all sensitive data to which she has access? Another example, where the employee found a USB flash drive in the washing room and want to return it to the legitimate owner. This user would open the USB drive on the computer to find out whom it belongs. However, this action would install the malicious software on the computer, which allows hacker to use credentials of the user to obtain sensitive information on its behalf, as the USB drive was intentionally left to use this type of employee. These were typical examples of insider threats, where current access control mechanisms RBAC,MAC,DAC (described in the Section 2.4.5) become powerless. For that reason, there is the need in additional countermeasures, which detect and prevent misuse of sensitive information before it become publicly announced by the media.

But before going in details of possible counter mechanisms, from the beginning of the chapter, we define concepts of insider, describe their main motivation and present

information about size of the threat, on the Section 3.1. This introduction should help to understand the main idea behind the relatively light-weight concept of insider mitigation, which we explain afterwards on the Section 3.3.

3.1 Insider Threats

One of the most important elements in every field of research is usage of common scientific language to describe problems and solutions. As the insider threat and data leakage research have been started relatively not so long ago, scientific literature presents a variety of definitions and characteristics of insiders. These characterizations often focus on different aspects of insider activity, which can be classified as technical, social or combination of both approaches together to studying insider crime [HCCY13].

Computer Emergency Response Team (CERT) has extensively researched the field of insider threat mitigation for the last decades and come to following definition of insider threats that captures both social and technical elements. A malicious **insider** is a current or former employee, contractor, or business partner, who meets the following criteria [SCM+12]:

- has or had authorized access to a network, system, or data of an organization.
- has intentionally exceeded or intentionally used that access in a manner that negatively affected the confidentiality, integrity, or availability of the information or information systems of an organization.

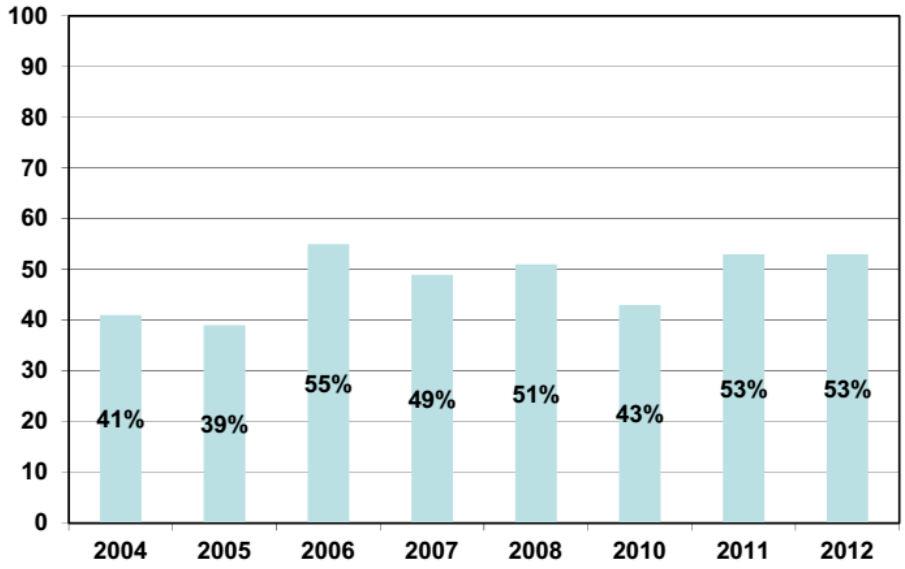

Figure 18: Percentage of participants who experienced an insider incident [Uni13]

This definition of a malicious insider is a good starting point for understanding why insider threats are so hard to prevent and detect. Insiders have a significant unique opportunity when it comes to committing an electronic crime over other types of intruders, as they can transparently bypass physical and technical countermeasures designed to prevent unauthorized access. For the reason, that the data of an organization must be available to them in order to perform business function at work properly. Moreover, they are not only informed about administrative and technological policies of the organization, but also about their weak points.

By the recent security survey conducted by CERT 2013 US State of Cybercrime Survey [Uni13], it was found, that half of respondents had experienced at least one deliberate insider incident in the past, the overall trend is presented in the Figure 18. The survey also revealed, that 53% of the respondents thought, that damage caused by insider attacks was more severe than damage from outsider attacks.

According to CERT, malicious insider activities are grouped into three major incident areas [CMT12]:

> **Information technology sabotage** - such incidents belong to a group of most technically sophisticated crimes and damage the most critical network, system or database elements otherwise their malicious actions would be easily repairable, and feeling of revenge over the abuser will not be satisfied. That is why typical person of such crime is system administrators, database administrators or programmers who have been recently demoted, fired or formally admonished. Attack is usually committed after they have left the organization from a remote place.
>
> **Fraud** - these crimes are motivated by financial gain and can continue during an extended period of time. Usually committed by customer services, help desk, medical, bank clerks. Having direct access to sensitive information they can offer it on the black market with the hope of monetary reward.
>
> **Theft of intellectual property** - usually committed by scientists, salespeople, programmers. As they believe, if they participate in the process of creating information they also keep ownership over it. They take this information with them as they leave the organization to start their own business or use it to obtain a higher position in a competitor organization.

Out of all insider cyber attacks, fraud and theft of data have been steadily growing in the last years. These types of attacks create another subclass so called data leakage. A ”data leak” refers to the fact, that sensitive data electronically leaves the organization boarders either accidentally or intentionally. The global trend of data leakage during 2006 and 2013 is made by InfoWatch [Cen14a] and presented on the Figure 19. Nevertheless, it reveals only a rough picture as only present the number of reported leaks,

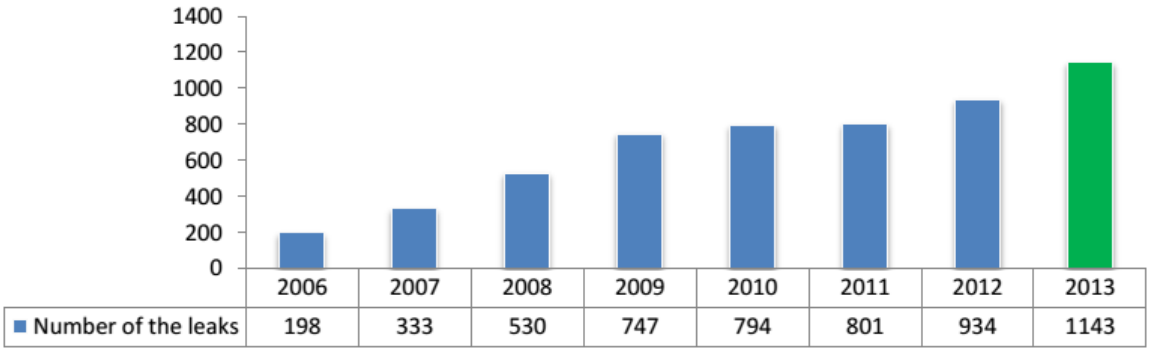

Figure 19: Number of registered information leaks, 2006-2013 [Cen14a]

while the real situation can be worse in several times.

Through incidents (see details in the Chapter 1) and observed trends over the past years, it has become clear, that many organizations are unprotected against serious leakage of sensitive data, due to the fact, that adequate prevention and detection controls are challenging to define and implement. Since, no single type of control is universally effective, defense in depth is required [McC08].

Further in the work, we address particular attention on data leakage problems and solution against them.

3.2 Data Leakage Countermeasures in DBMS

As the process of stealing data from DBMS for the authenticated user is a very straightforward. Attacker can retrieve a large amount of data even with a single-query. In the literature many methods to counter such attack vector exist.
Most approaches of data leakage prevention in DBMS are classified into two groups: syntax-centric and data-centric [MPNU10]. Syntax-centric approaches are based on analysis of user query expression while data-centric approaches are focused on what kind of data user tries to access and retrieve from the DBMS.
The analysis of existing methods to mitigate insider threat was conducted and those approaches were selected, which address the data leakage problem from completely different sides. Besides, the comparison of them with MVAL approach is presented in the Table 1 below.

DEMIDS approach

DEMIDS approach belongs to syntax-centric group, which was offered by Chung [CGL00]. They tried to create intrusion and insider detection system for relational DBMS. Consequently, this system is also applicable for data leakage detection. The approach derives

a typical user profile from DBMS audit logs. In details, the access pattern of user forms a "working scope" in respect to database schema and a given application. A "working scope" includes sets of attributes that are typically referenced together by primary or foreign key with some values. Also for proper work, DEMIDS requires a domain knowledge about the data structures and semantics of a given database schema [CGL00]. If the distance measure between "working scope" and a given user request is quite high, then DEMIDS considers such a user request as an anomalous. The shortcoming of this approach lies in the scalability. With increase of user number computation load for maintaining "user profile" also increases. Besides, there is always a need in domain expert when changes in database schema occurred.

Quiplet approach

The Quiplet approach was proposed by Kamra [KTB08] to overcome limitation of previous approach. It is based on analysis of SQL statement from the database log for further building normal behavior user profile. In order to build a profile, they convert every SQL query into basic data unit (a quiplet) that consisting from five fields (SQL command, projection relation, projection attribute, selection relation and selection attribute information) [KTB08]. Out of these fields, vector of features is built, that allows to apply learning-base mechanisms[1]. Afterwards, normal behavior user profile can be used for detection of anomalies behavior. They adapted this approach to RBAC system, as well to DBMS where no direct user roles exists. The weak point of the approach is the high false negative rate during the evaluation on the real database. However, the nature of the problem could lie in a particular data set.

S-vector approach

This approach belongs to the data-centric group. The author of it (Mathew [MPNU10]) also proofed, that it is more important to understand what data user retrieves from DBMS, then how the user expresses a query for retrieving this data. In this paper was illustrated, that even with significant modification of SQL query, it is possible to produce the equal result set. Consequently, most syntax-centric approaches are likely to mark those queries as anomalous, what lead to an increasing number of incorrect alarms. Besides, they offered their own data-centric approach [MPNU10], which outperformed the Quiplet approach. For every user of DBMS, S-vector approach creates a statistical "summary" of resulting data, representing it as a multi-variate vector of features with following statistical measurements like number of tuples and number of distinct tuples, media, mean, standard deviation of resulting data. It allows further to adopt statistical learning technics for clustering user query between normal and abnormal, in its turn, it is main drawback of the approach. Therefore, in case of database schema changing, S-vector must detect this changes and "user-profile" training should be done again, which is not always possible, due to continuous work of DBMS.

[1] In the work they applied Naive Bayes approach

MVAL approach

Barthel and Schallehn [BS13a] approached the data leakage problem from data quantity and quality side. They proposed to map sensitive data of DB to a MVAL, that allows to calculate a cost of data in the resulting relation as well as estimate the monetary risks for the organization per every user. Besides, approach defined suspicious and truncated thresholds, which should detect and protect against data leakage, respectively. This approach is very straightforward from point of understanding and computation complexity. We can not refer MVAL approach to qualitative type, although theoretically monetary cost of data correlates with its importance (quality) for the organization. However, the author calculated and truncated monetary value (MVAL) of result-set in tuple-wise way, even if the information in the tuple distinguishes fundamentally.

M-score approach

In the work of Harel [HSRE12], they proposed similar approach, but primary from side of data sensitivity. For the calculation of so-called Misuseability Weight (M-score) of tabular data. They combined in the model three factors: data quality, data quantity and the distinguishing factor (represents uniqueness of the data). Defining M-score of typical user request and comparing it with M-score of new user request are used for detection possible data leakage. The problem of this approach lies in the meaning of M-score, as it is not clear how to set threshold between normal result-set and misusable result-set. For example, extraction of few tuples with extremely sensitive data can have as high M-score value as a large number of tuples with low sensitive data. Even after applying normalization, you have some score from zero to one, but how to understand this value is not clear. Also, attacker can trick the M-score approach while sequentially quiring a DBMS for low sensitive data, which in the end can be merged together and reveal a lot of sensitive data overall. Another aspect, M-score approach has high-computation complexity, as it measures misuseability cost of every sensitive record in the result set.

After reviewing existing data leakage preserving approaches, there is no one golden standard, which of them to use against insider threats. Although, even within one group there are notable difference in the solutions. They all are capable to partially prevent data leakage. For that reason, we summarize properties which are critical in the production DBMS in the case of these approaches implementation:

Performance : an ideal approach must detect and prevent possible data leakage before the user request is fully executed. Other words, the time for detection must strive for the minimum. Besides, an approach must not greatly slow down the overall performance of DBMS.

Security simplicity: with this property, we address well-known security principle ”Keep Security Simple”. The majority of reviewed methods considers the

number of different features from query and from result-set to detect a possible data leakage, which leads to high complexity in understanding and configuration. Consistently, complexity could lead to error during implementation and further management in the production environment.

Data quantity/quality orientation : analysis of data quality requires to consider a significant number of parameters such as data relationship, uniqueness. Besides, they highly depend on the applicable domain. From the other side, quantity oriented approaches are focused only on the size of data. For that reason, they are much faster. Hence, there is always a performance trade-off between them.

	Data misused countermeasures				
	Syntax- Centric		Data-Centric		
Methods Properties	DEMIDS	Quiplet	S-Vector	M-score	MVAL
Performance					
Simplicity					
Data quantity oriented					
Data quality oriented					

Table 1: Comparison of methods against data misused problem

Previously discussed insider mitigation approaches were examined with respect to our properties. Table 1 uses the "traffic lights" color schema to illustrate how properties distinguish among approaches, where green color symbolizes the highest match to property, red - the worst and the white color means the absent of property.
Syntax-centric approaches (DEMIDS, Quiplet) are not primary focused on data leakage problems, but on broader type of attacks such as SQL injection, intentional data modification. For that reason, their efficiency to counter a data leakage is lower then data-centric approaches, which are specially focused to counter data leakage problem.
Combining data quality and quantity simultaneously, M-score needs to track significant number of parameters, which causes the performance issue. That is why this approach got red color in the Table 1. The S-vector requires additional "user-profile" re-training in case of database schema changing, that is why it looses in performance. The MVAL concept stands out from the others by its straightforward idea. Monetary value clearly explain the cost of data, that must simplify initial configuration and further management of the approach in the production DBMS. Moreover, due to a few parameters that influence monetary value of result set, MVAL approach has less computation overhead in comparison with others.

Next, we deeply look at the MVAL approach, as it helps a reader to understand how we integrate it into our application.

3.3 Leakage Data Preserving by MVAL Approach

Leakage data preserving by MVAL approach is based on calculation for every query user executes the total monetary value depending on queried tables and attributes as well as used operations.

In more details, every attribute $A_i \in R$ of a base relation schema R has certain monetary value ($mval(A_i) \in \mathbb{R}$). Knowing the monetary value of every attribute in relation we can derive monetary value of one tuple $t \in r(R)$ using the Equation 3.1.

$$mval(t \in r(R)) = \sum_{A_i \in R} mval(A_i) \tag{3.1}$$

Going deeper, the calculation of total monetary value of relation, the user query to database, base on the Equation 3.2.

$$mval(r(R)) = \sum_{t \in r(R)} mval(t) = \mid r(R) \mid * mval(t \in r(R)) \tag{3.2}$$

The Equation 3.1 represent the quantity of information per tuple, monetary value of all attributes in requested relation, and Equation 3.2 represents quantity of information per relation.
For calculation of monetary value of resulting relation we also need to consider type of SQL operation in the query and its resulting data. In the SQL, these are operations, which directly increase information content (such as JOINs, UNION), in this case the total monetary value should also increase. And another type of operations, which return single value that is calculated from values from columns. They are called aggregation (AVG, COUNT, SUM). Consequently reducing the number of tuples, these operations also should decrease the total monetary value of resulting data. In the paper [BS13b], Barthel proposed to used "uncertainty factor" parameter for this purpose Equation 3.3.

$$UF = \frac{1}{30} log_{10}(|val_{A_i,\ldots,A_k}| + 1) \tag{3.3}$$

Also "uncertainty factor" is used when resulting data contains record with NULL values. From standpoint of the monetary value, NULL does not always mean lack of value. As knowing that there is NULL value for certain record can lead for possible information gain. Afterwards, the monetary value of a query as well as several queries of one user is compared to suspicious and truncate thresholds within a specific period of time.

Suspicious threshold: $thr_{susp} \in \mathbb{R}$ is used to identify the potential misuse of sensitive data. This threshold is set in a way to allow user perform their duties without exceeding

it. When total monetary value exceeds this threshold the suspicious message must be sent to alert log for further investigation and taking appropriate measures.

Truncate threshold: $thr_{trun} \in \mathbb{R}$ is used to identify evident malicious activity directly. When an accumulative monetary value exceeds this threshold, the DBMS must truncate the output result of queries to the monetary value of the threshold and truncate message must be sent to alert log for taking further administrative action.

Alert log: The alert log represent system that captures every exceeding of aforementioned thresholds, writes them continuously and allows a tracing of violations overtime. Responsible person periodically analyses it and inform responsible security official to take further action.

Since we measure how much information was extracted over a period of time, the accumulated monetary value of a user is reseted periodically (hourly ,daily ,weekly). It is also recommended [BS13a] to perform calculation of total monetary value before the physical data query to decrease unnecessary load on DBMS. As, every modern database engine contains meta data tables that can be utilized for calculation a resulting monetary value before the actual data from database is retrieved. If it exceeds truncate threshold, we limit the result set to a maximum releasable tuples ($tpl_{max} \in \mathbb{N}$) for a certain result relation $r(R)$ by Equation 3.4:

$$tpl_{max}(r(R)) = \left\lceil \frac{thr_{trun}}{mval(t \in r(R))} \right\rceil \tag{3.4}$$

Truncation is only applied on operations that present result to the user, all other activity like, DBMS maintenance, changing data are executed omitting monetary value violation concept.

On the Figure 20, we shows how monetary value violation is positioned themselves in the security defense model of typical DBMS, this allows to implement so call defense in depth (also known as deep or elastic defense). The most important aspect of it is [Byr08] :

- **Multiple layers of defense** - We must not rely completely on a single point of security, no matter how perfect this countermeasure is.
- **Differentiated layers of defense** - We must be sure, that each of the security layers is more or less different from others. This ensures, that in case an attacker can find a way to pass some layer, he/she does not have the magic key for getting through all the subsequent defense layers.
- **Context and threat specific layers of defense** - Each of the defenses should be designed to be context and threat specific.

Monetary value violation layer located beneath the access control layer, it extends access control layer with additional restriction rules. On the opposite side of access control, it is common practice to define views provide content-dependent security [Wil88] and give users access to a specific portion of data without having direct access rights. The last layer is encryption [SVEG10], because it uses different sort of encryption [SVEG10] DBMS, file system, application or client side to protect the raw data.

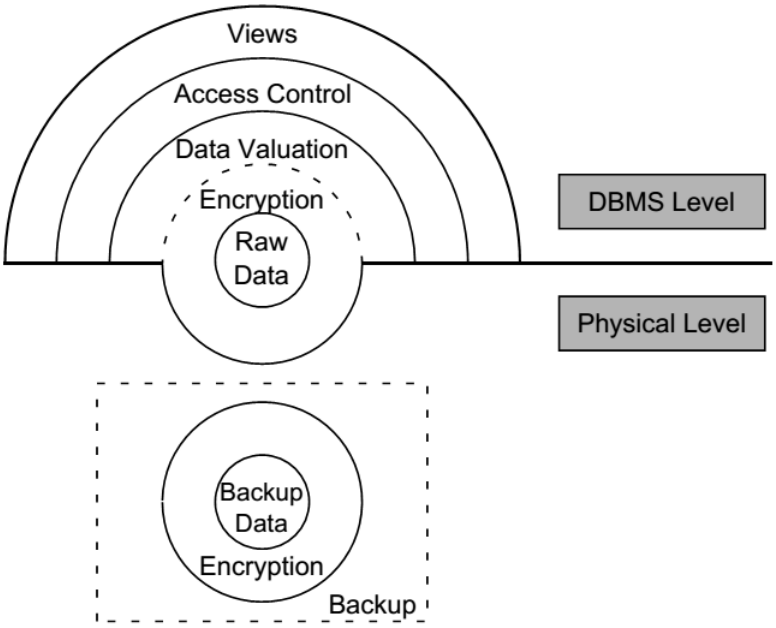

Figure 20: Security defense model of DBMS and physical level [BS13b]

3.3.1 Data Definition Language Extension

Realization of this concept requires additional Data Definition Language (DDL) commands (Section 2.3.3) to manipulate truncate and suspicious thresholds, as well as monetary value for every attribute of a relation. The example on the Listing 2 illustrates syntax, how the user can define define both thresholds together with monetary values for each attributes during creating a table. In the code example, the thresholds parameters are defined as system variables, monetary value for attribute 1, 2 and 3 are defined directly, while for attribute 4 is set default behavior. Default monetary value for non configure attribute must be always equal zero ($mval(attribute_4) = 0$). Such behavior of default rules in DBMS with implemented leakage data preserving layer allows it to work like a traditional non-modified DBMS, in case the DBMS administrator did not defined any rules. The similar set of rules are applied for changing the existing database are presented in the Listing 3.
First command add additional attribute to existing table and second one modify already existing attribute, set new monetary value to 1 on the attribute 3.

```
suspicious_valuation=2000;
truncate_valuation=4000;

CREATE TABLE table_1
(
        attribute_1 INT PRIMARY KEY MVAL 0.1,
        attribute_2 UNIQUE COMMENT "important" MVAL 10,
        attribute_3 STRING MVAL 0,
        attribute_4 DATE
);
```

Listing 2: DDL monetary value creation example [BS13a], modified

```

ALTER TABLE table_1 ADD attribute_5 STRING MVAL 2;

ALTER TABLE table_1 MODIFY  attribute_3 STRING MVAL 1;
```

Listing 3: DDL monetary value altering example

After we finished detailed description of MVAL approach, the possible improvements are presented below.

3.3.2 Further Extension

In this subsection we present points, which Barthel and Schallehn did not cover in the papers [BS13b, BS13a] and offer their improvements.

The problem appears when a user repeatedly accesses the same data. For better problem understanding, let us consider the example in a bank. Client calls to the bank to increase daily limit transaction to buy a costly product. The clerk authenticates the client by the phone, comparing the retrieved information from DBMS with answer of the client. From that time, accumulated monetary value of the clerk starts increasing. However, if the same client calls later to return daily limit transaction on the card to the previous value, the clerk needs to retrieve again the same information about client, this operation again increases accumulated monetary value. Another example with a human factor, when a user of DBMS accidentally refreshes screen in the browser, that leads to execution of repeating query on the DBMS.
From the examples above, it is possible to increase accumulated monetary value just repeatedly accessing the same data and active suspicious and truncation threshold. To answer this problem, we offer to define a new ”retrieved data expire” interval which allows in case of similar retrieved result from DBMS do not consider them in the calculation of accumulated monetary value.

Another question, that we found, belongs to problem of inferential control (more details about it in the Section 2.4.6). The idea of it lies in the possibility to perform sensitive

data disclosure via non-sensitive data. To demonstrate this problem we present table with sensitive data about workers (Table 2). The top tuple represent monetary value of every attribute. "Id" and "name" attributes have monetary value of 0 and "salary" attribute has monetary value of 10, it means, that "salary" attribute is sensitive data.

Monetary Value		
0,0	0,0	10,0
id	name	salary
1	Yana	3100
2	Natali	2100
3	Mirella	2500
4	Thomas	2900

Table 2: Represent worker table of an organization

```
1 SELECT id, name FROM worker WHERE selary > 3000
```

Listing 4: Example of SQL query to show inferential problem of MVAL approach

Executing SQL query in the Listing 4 over the Table 2, allows the attack to retrieve the name of worker with salary higher then 3000. As the projection parameters of this SQL query (Listing 4) does not contain sensitive attribute the monetary value of the result set will be equal 0. Sequentially executing such SQL query and changing only WHERE condition, an attacker can find out what is the salary of every worker in the organization. This problem exists not only in WHERE clause, but also in the variation of JOIN clause (INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN) as JOIN contains condition statements. To solve this problem, we offer while monetary value calculation of result set also to consider the monetary value of every sensitive attribute in the condition statement, which did not appear in the final result set.

In previous chapters we described overall information about databases, their design, architecture and traditional measures for their protection. In addition, the technology overview of used tools was done as well as the detailed information about insider threats and countermeasures were given.
Further, we proceed to the design of the demonstrator application.

4. Design of Demonstrator Concept

For the purpose of studying a data leakage preserving by MVAL approach (see details in Section 3.3) as well as presenting its work, we design and implement a web based demonstrator platform. It clearly exhibits the main properties of concept and can be used in the future as a core for further developing and extending functionality of MVAL approach. Besides that, it allows, in case of necessity, to add new insider mitigation techniques for demonstration purposes.

At the beginning of every developing, we plan and defined main requirements which the developing application must support. Based on them we choose technology and algorithms for the realization. At this chapter we begin with defining overall architecture of the application, then introduce core milestones of graphical interface. At the end we describe functionality of developing DBMS engine with support of data leakage preserving extension and present conceptual model of DB which data is used for demonstration.

4.1 Architecture Requirements

For the realization of application, we have been choosing between two conceptually different application architectures native and web based. Native applications are installed on a personal or work computer, traditionally it is desktop or laptop. Web applications are installed somewhere in the internet and to access them you need any computer device with installed web browser. Despite both types of applications can solve identical tasks, they have fundamental difference.
As every native (desktop) application has following limitation:

> **Update challenges** - every update to an application must be applied by the user directly. That could lead to possible error in the application in case of not following update instructions.

Access problem - native applications are only can be run on the device where they were installed. It means, without direct access to the device it is not possible to run a native application.

Cross-platform problem - native applications are developed only for a particular Operating System (OS), therefore they will not operate on devices with different OS.

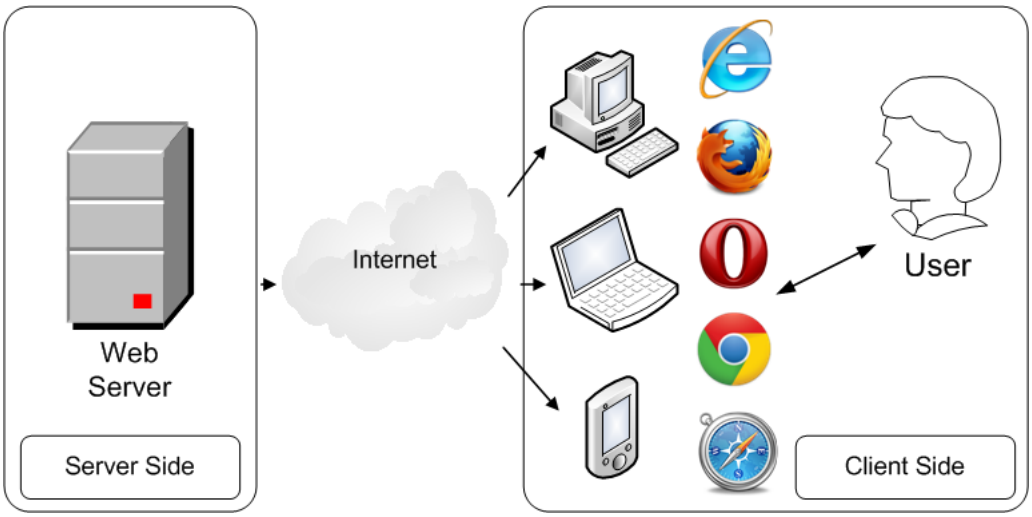

Figure 21: Web based application architecture

Due to theses limitation, we have decided to use web based architecture (see Figure 21), as it has following advantages:

Cross-platform - unlike native applications for specific platform (Windows, Linux, Mac OS), one web application works across all platforms and OS, both now and in the future as long as a web browser exists. Additionally, even if another OS appears in the next few years the same web application will work on that platform as well.

Future-proof - in a world of fragmented OS, with hundreds different languages and API, only a web is constant. It is shared by all platforms and OS and it is for certainly will not disappear in a near future.

Maintainability - as one web application works on every platform, future maintenance is simple. All changes made to that single application instantly reflect across all platforms.

Auto updates - since web application runs in a browser and is not installed on the device itself, any updates to it, instantly reflect to all users. Users do not need to install the latest update or do anything at all.

Accessibility - unlike traditional applications, web application are accessible from any part of the world and at any given time. User only needs computer device (laptop, tablet, smart phone) with decent browser and internet connection. As a result, a user is in charge to choose where and when access the application.

Considering aforementioned properties, we can easily deliver application to broader user audience and from the start user can concentrate on understanding data leakage preserving approach instead of installing, configuring required components to run the application.

In typical web applications, there are client and server involved. A client is a web browser, like Internet Explorer, Google Chrome, Firefox (see on Figure 21). The server is a web application server at a remote location that receives, processes web requests and send result pages to the end client.

Another aspect of web applications is how the logic code is processed either on server-side or on client-side. Comparing two approaches, we decided to develop the demonstrator, which is based completely on client-side processing, as the server-side processing has following difficulties:

Network connectivity problem - due to all processing made on the server, after each interaction, the user must always have permanent connection to the server.

Server maintains problem - as the server is processing all code, the necessary complicated running environment must be configured (programming language with libraries) and maintained, during the whole life time of application.

Security problem - due to complicated server configuration, it becomes possible to execute other attacks like SQL injection, remote code execution attack.

Client side code processing architecture solves aforementioned problems, but in addition to that has following advantages:

Faster response times - user must not wait for the server to process the request and send the page back.

Interactivity - allow more interactivity by immediately responding to actions of a user, due to the code is executed on the device of user and not on remove server.

Less overhead on the web server - transferring processing task to client side, there is no necessity in powerful server hardware to handle hundreds of clients.

Among different client-side programming languages, the preference was given to JavaScript languages (detailed description in the Section 2.5.1), one of the implementation of ECMAScript. It is de-facto web standard [Sur14a, ZZ04], default scripting language for all popular browsers and does not require to install additional software frameworks like in case of using other technology for rich internet application as Adobe Flash, JavaFX or Microsoft Silverlight.

4.2 Development Environment

For developing of the application, we used Eclipse Kepler [EF14] version 4.3, due to following main features:

- free of cost - there is no need to pay for the usage of integrated development environment.
- code completion - Eclipse assists programmer during writing the code, instead of looking through documentation programmer can choose appropriate function or variable from the drop list.
- syntax checking - helps to write correct code while typing.

It is multi-language software development platform written primarily in Java comprising an integrated development environment and a plug-in systems. To extend support of used stack of web technologies (HTML, JavaScript, CSS), Web Tools Platform [TWH05] was installed.

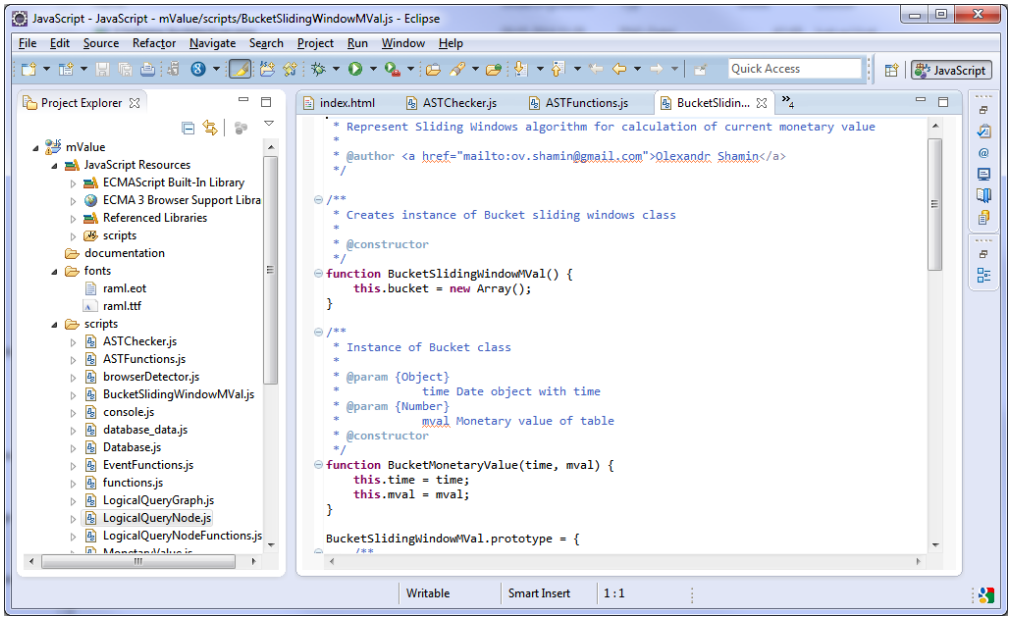

Figure 22: Eclipse : integrated developing environment of the demonstrator

4.3 Graphical User Interface Requirements

Graphical User Interface (GUI) connections user with an application, as it gets requests from the user and gives responses backwards. During designing the GUI of demonstrator we want to enchant the final usability of the application.
Usability is not a new concept in the field of human-computer interaction. Moreover, it was clearly defined in ISO standards (ISO/IEC 9126-1 [ISO98], ISO 9241-11 [ISO00]). In the ISO 9241-11 international standard, they provide guidance on usability and define it as: "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [ISO98]. The standard subdivides "usability" in the following sub factor:

Effectiveness - with which accuracy and completeness the user achieve predefined task.

Efficiency - how many resources (time, actions) user does need to accomplish predefined task and how many mistakes user performed while completing it.

Satisfaction - a ratio of pleased opinions about the completed action from the viewpoint of the user.

Besides, ISO 9241-11 standard highlights, that usability is context depended. It means, that the physical environment, type of user, tasks, equipment (computer, tablet, phone) influence on the level of usability in the final product.

To address effectiveness, efficiency and satisfaction factors, we properly redefine the "Eight Golden Rules of Interface Design" formulated by Shneiderman in his book [SP05] and implement them in the GUI of application. These below listed rules derived heuristically from experience not only him but also other human computer interaction specialist and applicable in most interactive systems.

1. **Strive for consistency.** Primarily refers to common sequences of action is applied in similar situations. As well as identical terminology within prompts, menu, help screens and consistency in colors, fonts, capitalization are employed within an application. Consistency is a strong determinant of interface success [SP05]. Exceptions out of this rule should be perspicuous and limited in number.

2. **Offer informative feedback.** Every action of a user in the application should have an appropriate feedback. Of course, some frequent actions like mouse click on non-interactive part of the interface do not require any feedback, whereas, for infrequent and significant actions, the response should be substantial. Implicit feedback allows to minimize errors and complete tasks faster because results on the action are observable before an operation is accomplished.

3. **Design dialogs to yield closure.** Every series of actions should be organized in small groups with implicit beginning, middle and end phases. Information feedback should indicate complication of group of activities and gives user feel of the fulfilled their duty, in a sense of relief and prepares the user to the next group of actions. A good example, when user electronically orders flight ticket, the website carries user through steps of selecting ticket to checkout, finishing with a clean information page that indicates finish of the transaction.

4. **Prevent errors.** The interface should be designed, in a way that prevents a user to make a significant error. Moreover, in case the user makes errors, it should detect them and provide a simple, helpful recovery solution. For example, in the web forms user is asked to fulfill zip area and in case of incorrect input, the system should inform about erroneous field and make a suggestion to correct the error based on other fields like address and city.

5. **Cater to universal usability.** Designer should go beyond of "typical" user. Universal usability means, that users can be of different ages, experience levels and physical limitations. A good example is a user, who uses an application frequently, requires possibility to increase the pace of interaction. Therefore, adding for often used functions shortcuts, functional keys and macro facilities is a very helpful move.

6. **Permit easy reversal of actions.** As much as possible, user actions in the application should be reversible. What is not always possible due to administrative or application logic constrains. However, this feature relieves anxiety and stimulate for exploration of unknown options, as the user knows that mistakes can be undone. Reversibility unit might be either a single action over data entry or a complete sequence of actions.

7. **Support internal locus of control.** Refers to the strong believe that the user can control the interface and that the interface directly responds to the user actions. Therefore, any changes in familiar behavior or surprises of the application are not acceptable. As well as any difficulties in obtaining the necessary information or inability to produce desired results.

8. **Reduce short-term memory load.** Due to the limitation of human in short-term memory, designers should avoid complicated interfaces with an overwhelming number of options and frequent windows changing. Hence remembering them requires a considerable amount of time and at the beginning of the learning phase leads to errors. As a result of it, keeping design of applications as simple as possible is beneficial.

The most problematic rule is universal usability, as solving it requires a considerable amount of time. For that reason, we target the graphical user interface for users, who traditionally closely works with DBMS (database administrator and programmer). Besides that, we also target application on the resolution of the device's screen and define

it to 1024x768. This does not mean that the application is not possible to run at a lower resolution, but in this case usability suffers greatly.

At this step the GUI requirements are defined, and we begin considering the functional requirements of DBMS engine.

4.4 Functionality Requirements of DBMS Engine

For our demonstrator, we plan to realize only these parts of DBMS functionality which are essential for presentation data leakage preserving concept [BS13a]. All other important parts of traditional database are omitted (multi-user support, extensional query optimization, data description language and so on), because their presence in the developing application only move point of attention from what we want to demonstrate and force us to spend considerable amount of time on irrelevant tasks.

4.4.1 Query Processing

The core functionality for DBMS engine in this work is SQL query processing. It allows us not only to show some predefined queries with results, but also gives users the ability to interact with demonstrator and check how this concept could work, when it was implemented in production DBMS.

For the further explanation of details of the query processing we defined the SQL query on the Listing 5. And based on it we present planned internal structures of the developing application during designing phase.

```
1 SELECT * FROM talbe1
2 JOIN table2 ON table2.attr1 = table1.attr2
3 WHERE table1.attr3=555 AND table1.attr2 = "SSS";
```

Listing 5: Used SQL query for demonstration of concept

The overall SQL queries processing concept of DBMS engine is presented on the Figure 23 and consists of three phases: query compile, query optimization and query execution.

Execution of human written text is not trivial task, even if it is structured by predefined sort of rule like in case of SQL. For that reason SQL query has to be converted from human readable form into computer recognizable form. Therefore, the process of SQL query processing starts from paring of entered SQL query, during this step we identity the query tokens, such as SQL keywords, attribute names, relation names that appear in the body of the query.

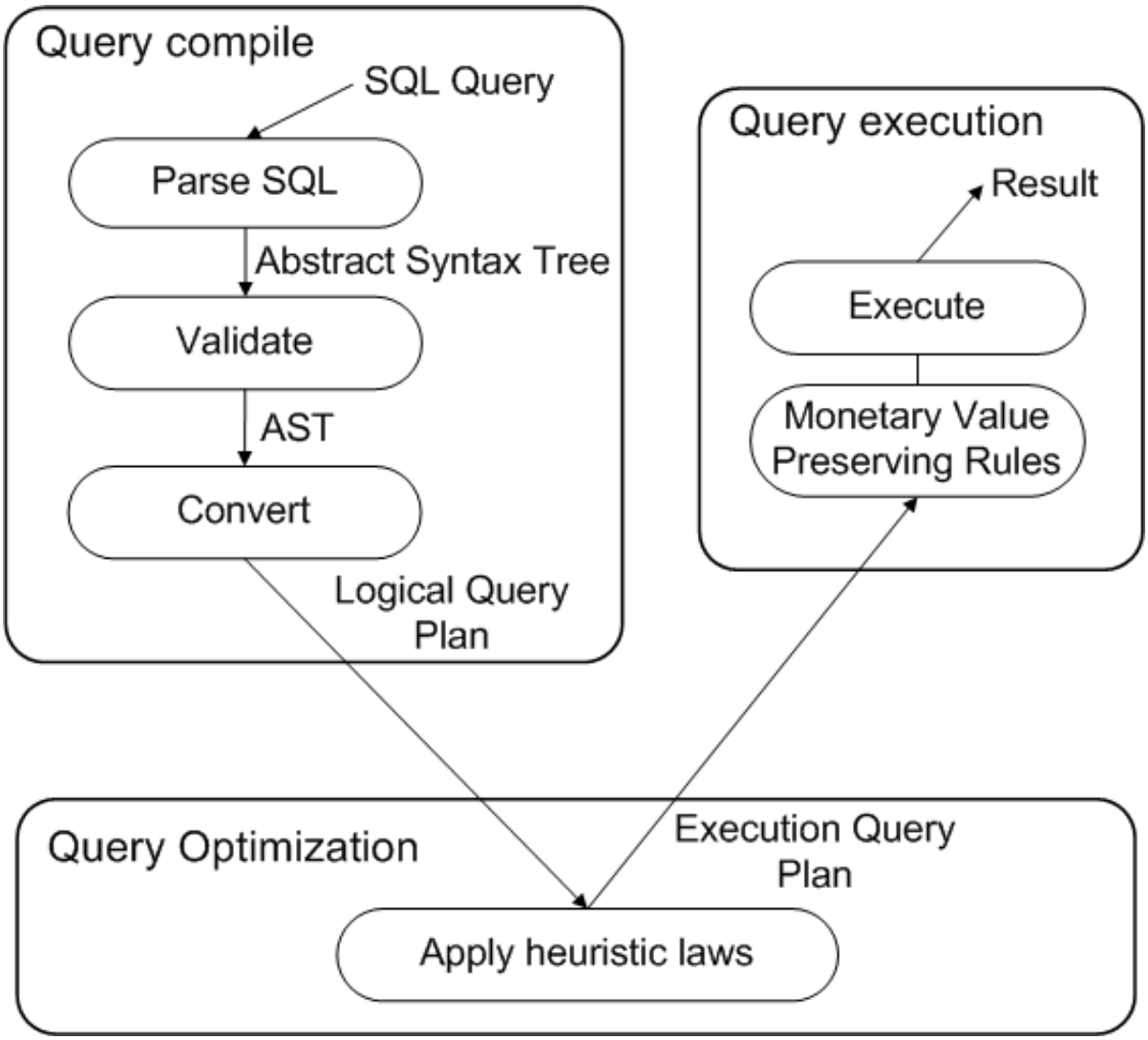

Figure 23: Query processing schema in developing DBMS

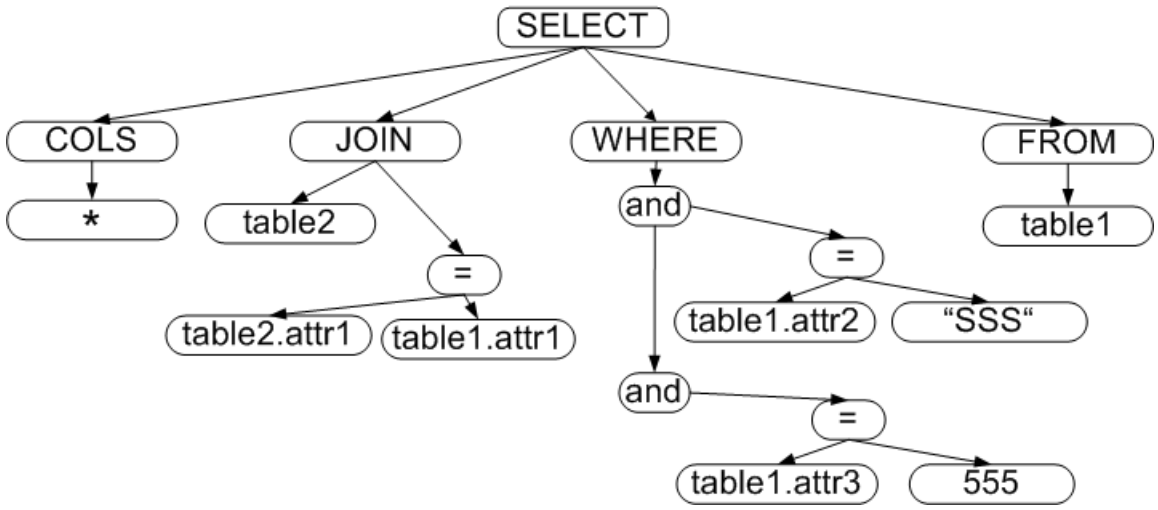

Figure 24: Abstract syntax tree of the query Listing 5

Afterwards, we check the syntax of the query to determine whether it is formulated according to the syntax rules of SQL. The result of parsing step is Abstract Syntax Tree (AST), illustrated on the Figure 24, where each node has to represent key elements of SQL query.

At the next step we validate AST, that all attribute and relation names of a queried database are valid and exists. If an error is found, we have to provide a small explanation what is wrong with initial query to allow user fix the error. In case if AST is valid we build internal representation of it, as a tree data structure, in our work we call it Logical Query Plan (LQP).

Usually query has many possible execution strategies, the phase of improving executing performance and choosing the fastest strategy is called a query optimization. As our application runs on the client side, inside of the browser, full database is stored also in the internal virtual memory of a browser. Through location of database in memory any issues with access to slow data storage as hard drives are excluded, that is why we decided to simplify query optimization step and only implement following heuristic rules:

- **Push down selection** - reduces number of tuples.
- **Push down projection** - reduces number of attributes.
- **Push down joins** - perform most restrictive join operations before other joins, reduce number of tuples.

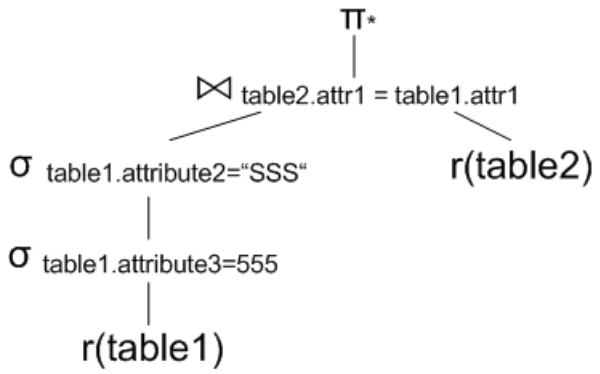

Figure 25: Representation of LQP as query graph

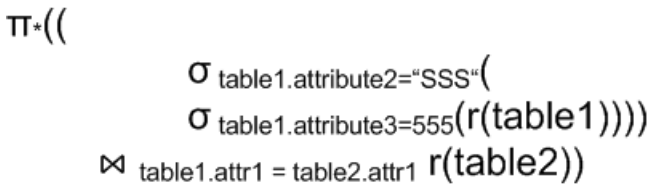

Figure 26: Representation of LQP in relational algebra operations

As a part of the demonstrator we also plan to present internal LQP after optimization phase to the user in the way of query graph (Figure 25) and relational algebra expressions (Figure 26).

4.4.2 Internal Functionality

As SQL language was designed for managing data in a relational DBMS (Section 2.3.1). The support of SQL requires from us to implement following set of basic relational algebraic operation (Section 2.3.2):

- **Select** - used to filter a subset of tuples from an original relation that satisfy a selection condition, denoted by σ symbol.
- **Project** - used to select specific columns from a relation and discards other one, denoted by Π symbol.
- **Union** - allows to unite two relations into one resulting relation that contains all tuples from both original relations, denoted by $\cup$ symbol.
- **Rename** - allows to rename output relations after applying other relational algebra operations, denoted by ρ symbol.
- **Division** - returns resulting relation that contains tuple presented in the first relation that does not contain in the second relation, denoted by $-$ symbol.

- **Cartesian Product** - returns one relation, that contains all possible combined tuples from two two relations, denoted by $\times$ symbol.

These operations build a complete set, it means that any of the other relational algebra operations can be expressed as a sequence of operations from this set. As an example, both derived intersection and join operations are presented on the Equation 4.1, Equation 4.2 accordingly.

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R)) \tag{4.1}$$

$$R \bowtie_{condition} S \equiv \sigma_{condition}(R \times S) \tag{4.2}$$

Derived operations :

- **Intersection** - returns resulting relation, that contains tuples common to both original relations, denoted by $\cap$ symbol.
- **Inner Join** - returns resulting relation, that contains those tuples from both original relations which satisfied join condition, denoted by $\bowtie$ symbol.
- **Left Outer Join** - returns resulting relation, that contains all tuples of the "left" relation combined with tuples from "right" relation using the join condition. If the "left" tuple is not satisfied the join condition, then its attributes are filled with NULL values. This operation is denoted by $\bowtie$ symbol.
- **Right Outer Join** - works similarly to left outer join, but for the "right" relations, denoted by $\bowtie$ symbol.
- **Full Outer Join** - combines effect of using both right and left outer joins, denoted by $\bowtie$ symbol.

In addition to relation algebra operation, we plan to implement SQL comparison operation (see on the Table 3) and logical (AND, OR) operators. We used them in WHERE and JOIN condition statements to extract only those records that fulfill a specified criterion.

Using aforementioned operations, we can build different variants of SELECT statements, like most well know **SELECT-FROM-WHERE** query and their modification with different JOIN operations. We also plan to implement **ORDER BY** clause, that allows the user to order the tuples in the result of a query by the values of one the attributes and **LIMIT** clause to limit query results to those, that fall within a specified range.
By the end of the work, our DBMS engine has to process following queries, see example on the Listing 6.

Operation	Description	Example
=	Checks if the value of two operands are equal, if yes then condition becomes true	(a = b) is false
!= , <>	Checks if the value of two operands are not equal, if value is not equal then condition becomes true	(a != b) is true (a <> b) is true
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is false
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true	(a < b) is true
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	(a >= b) is false
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true	(a <= b) is true
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true	(a !< b) is false
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true	(a !> b) is true

Table 3: Planned SQL comparison operations support

```
1  SELECT rel1.attribute_1, rel2.attribute_2, rel3.attribute_3
2  FROM rel1, rel2
3  RIGHT JOIN rel3 on rel1.attribute_1 = rel3.attribute_1
4  WHERE rel1.attribute_1 > NUM_VALUE AND rel2.attribute_2='STR_VALUE'
5  ORDER BY rel1.attribute_1
6  LIMIT 2,3;
7
8
9  SELECT rel1.attribute_1, rel3.attribute_3
10 FROM rel1
11 RIGHT FULL JOIN rel3 on rel1.attribute_1 = rel3.attribute_1
12 WHERE rel1.attribute_1 > NUM_VALUE AND
13     ( rel3.attribute_2='STR_VALUE' OR rel3.attributre_3 = NUM_VALUE )
14 ORDER BY rel3.attribute_1;
```

Listing 6: Planned example of SQL queries support

Query in SQL can be very complex and in this work we do not plan to support most of operators and functions from production DBMS, but we provide ability for flexible extension of internal functionality.

4.4.3 Integration of MVAL approach

Data leakage preserving by MVAL approach (details in the Section 3.3) has to be integrated into the phase of query execution. Illustration is presented on the Figure 23, it has to be integrated in the last step of query processing schema of our DBMS engine. As it has to change the output relation in case of monetary violation of resulting data. Beside that, demonstrator has to provide ability to change global monetary value parameters, truncate and suspicious thresholds (Section 3.3), as well as monetary value for every attribute of the presented relations and displays the current accumulated monetary value to the user. In case of violation the monetary value rules, the alert log (details in Section 3.3) has to show messages about occurred events with small explanation what happened.

For calculation of accumulative monetary value in the work, we use the Equation 4.3. It summarizes monetary values of all resulting relations within certain interval.

$$acmval(interval) = \sum_{i \in interval} mval(r_i(R)) \tag{4.3}$$

Interval parameter has to be a user configurable. And its time scale has to vary from from seconds to days.

For resetting an accumulative monetary value in the work, we considered two different methods. First method is presented by Barthel [BS13a] in the description of MVAL approach and it is based on resetting an accumulated monetary value after fixed interval. And another one we present within this thesis, it is based on resetting accumulated

monetary value within sliding fixed time interval. On the Figure 27, we demonstrate the work of both methods and how an accumulated monetary value is calculated on the identical working behavior.

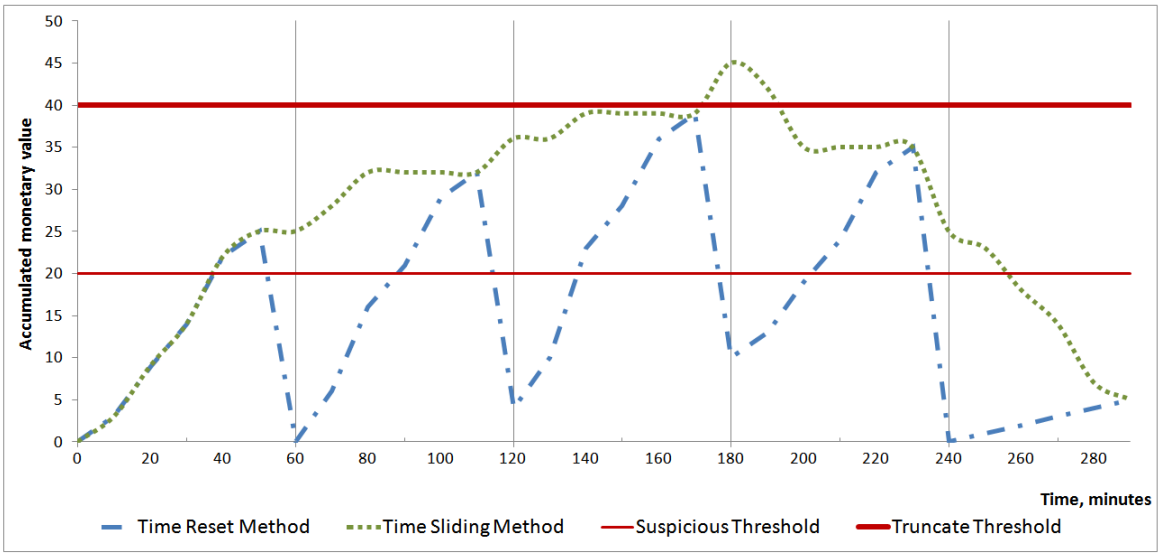

Figure 27: Comparison of two methods for calculation of accumulated monetary value

For the comparison of them, we use 60 minutes reset interval. The time sliding method (blue dash-dot line on the Figure 27) is more sensitive and restrictive, because an accumulated monetary value exceeds the truncate threshold while fix time reset methods is always below the truncate threshold. As a result of this comparison, we have to implement time sliding method in our work.

At this step we finish the design phase of demonstrator application, we defined requirements for the running environment, graphical user interfaces as well as requirements for DBMS engine with support of data leakage preserving by MVAL approach.

4.5 Database Schema Design

In order to present work of DBMS engine and underlying monetary value preserving concept (see Section 3.3), we design a database of hypothetical university and fill it with data. Logical schema of the database is presented on the Figure 28 in the form UML diagram.

It consists of four relations:

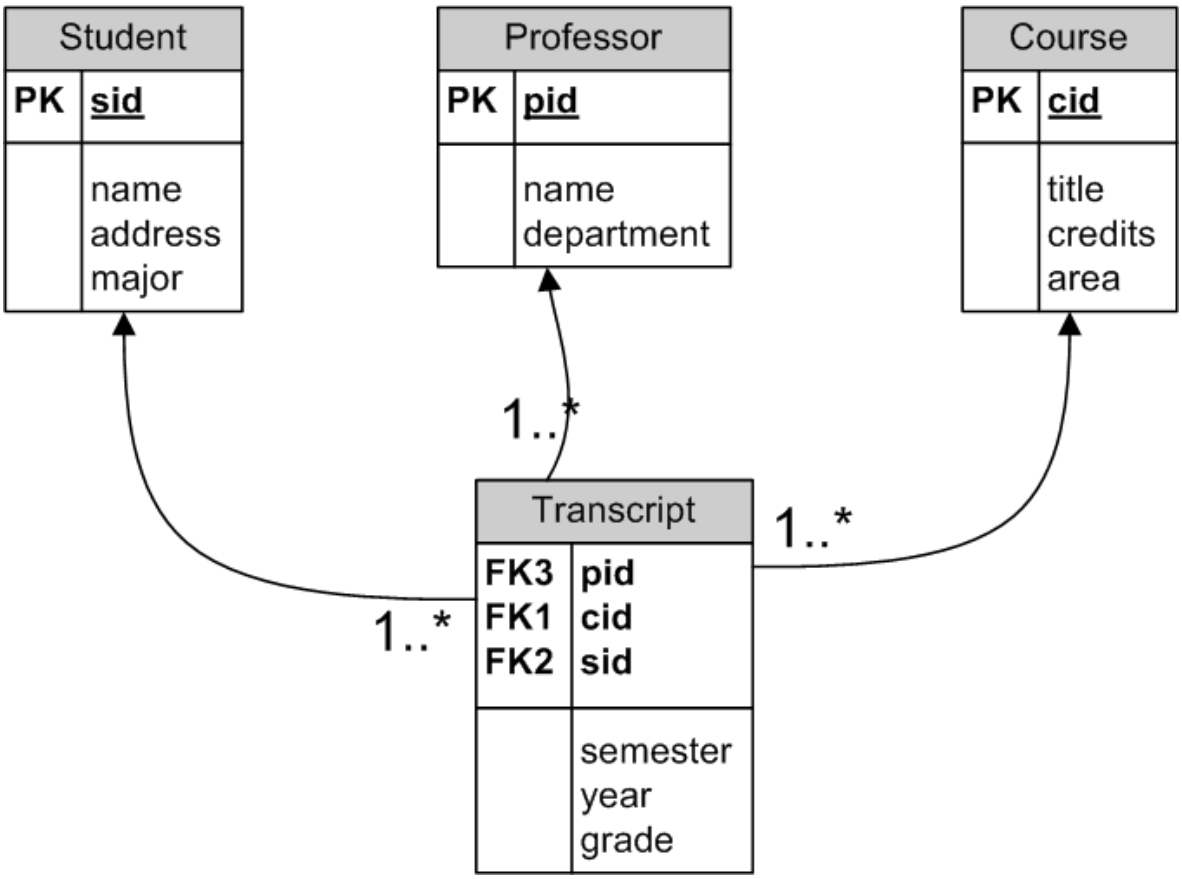

Figure 28: UML model of database prototype

1. **Student** - contains information about student of hypothetical university, with name, address of residence and faculty of university where he/she belongs. Student[sid[1], name, address, major]

2. **Proffesor** - stores information about teaching staff of hypothetical university, with name of the teacher and deportment of university where he/she belongs. Professor[pid, name, deparment]

3. **Course** - stores information about teaching courses of hypothetical university, with title of the course, number of credits and to which area course belongs. Course[cid, title, credits, area]

4. **Transcript** - contains information result of student passed exam, identificator of the student, course identificator, identificator of the professor who took the exam, the period when exam was taken and mark. Transcript[sid[2], cid, pid, semester, year, grade]

[1]Underlined line means it is a primary key
[2]Double underlined line means it is a foreign key

We intentionally design straightforward database schema to allow a user quickly catch the relationship among tables, consequently, encouraging for the use of complex SQL query. The ability to change logical schema as well as data within tables are not planned within this work, as their presence is not essential for presentation data leakage preserving by MVAL approach.

5. Implementation

In this chapter, we describe the implementation of the demonstrator for data leakage preserving by MVAL approach. The detailed overview of the design was presented in the previous chapter. Implementation of the demonstrator is divided into several large modules: Graphical User Interface, query processing and data leakage preserving by MVAL extension. Only connecting them together, we can clearly illustrate the work of concept.

At the beginning of implementation we tried to find already developed JavaScript DBMS engines based on our requirements (Section 4.4). However, existing open JavaScript implementations realized only simple storing and extraction functionality over a group of records without support of SQL. Their adaptation to the requirements of demonstrator would demand from us to spend huge amount of time for understanding their internal implementation for the further redesigning. For that reason, we decided to developed JavaScript DBMS engine from scratch, but with help of third party libraries, because full development would exceeds the scope of this work as the primarily aim is to demonstrate the work of data leakage preserving by MVAL approach.

5.1 Architecture Implementation

The overall architecture for our web application is presented on the Figure 29. We used the server as place for storing files of the demonstrator and distributing them among clients. All other functionality such as SQL query parsing, processing, accumulated monetary value calculation, presentation of relational algebra expressions and GUI rendering is shifted to the client.

As the web server functionality requirements were deliberately limited only for storing and redistributing files without any further processing. It allows to select a web server

from dozens different realization (Apache Server, Internet Information Services, Nginx and so on).

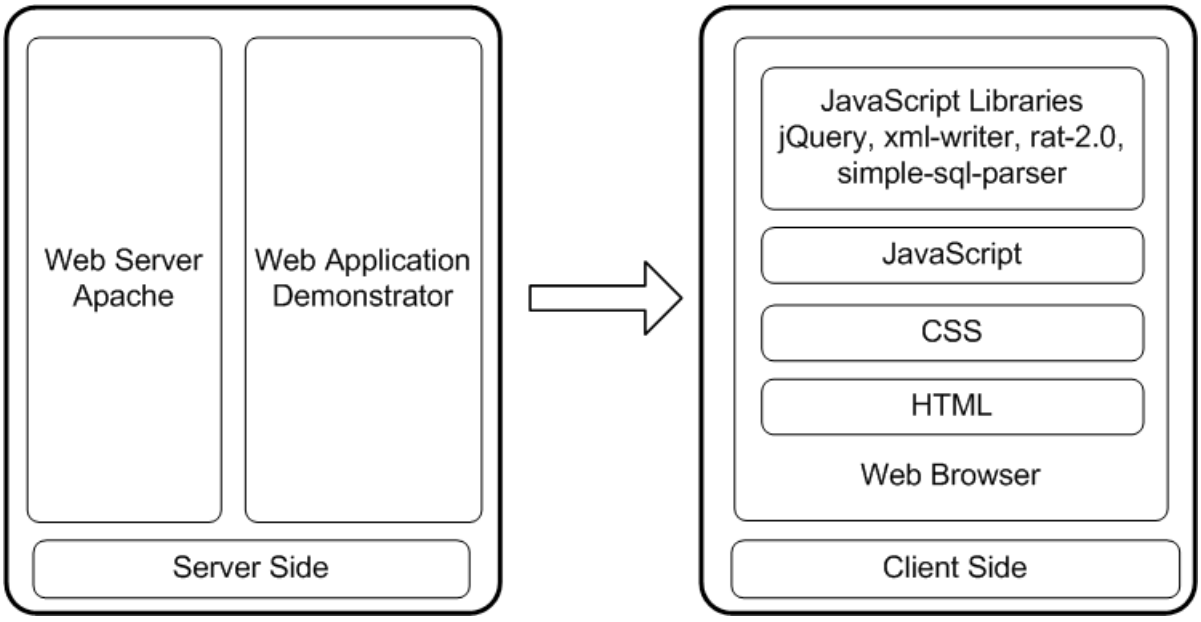

Figure 29: Architecture of the demonstrator application

In our case we have chosen Apache Server, as it has following advantages:

Openness - the web server is an open source application. The developer can easily apply own code to improve internal functionality.

Cost - as it is open source, user must not pay anything for the usage.

Portability - web server Apache can be run on every modern operation system platform like Linux, Windows and Mac OS.

Features - due to open architecture, the web server contains a wide range of modules. Which extend the basic functionality with support of programming languages, encryption connection between user and web server and so on.

The application is designed and built using modular programming technique. That enforced us to separate functionality of the application into independent modules, where each of them contains all necessary information to execute only particular aspect of the required functionality. Besides that, implementation of modular programming techniques in the application recoups in the future when the need to extend functionality appears.
The UML model of demonstrator is presented on the Figure 30. It contains many packages, but they can be grouped into three main groups:

DBMS packages - contains code responsible for functionality of developing DBMS engine (more information about requirements in the Section 4.4). In addition to that, it contains code with stored database (logical schema of it presented in the Section 4.5) for the demonstration of monetary value preserving concept.

Relational Algebra Presenter packages - contains code for converting logical query plan of SQL query into understandable relational algebra expression in form of XHTML and query graph in form of SVG for further presenting inside the application.

GUI packages - contains code for handling user inputs, interaction with user, help and error handling system.

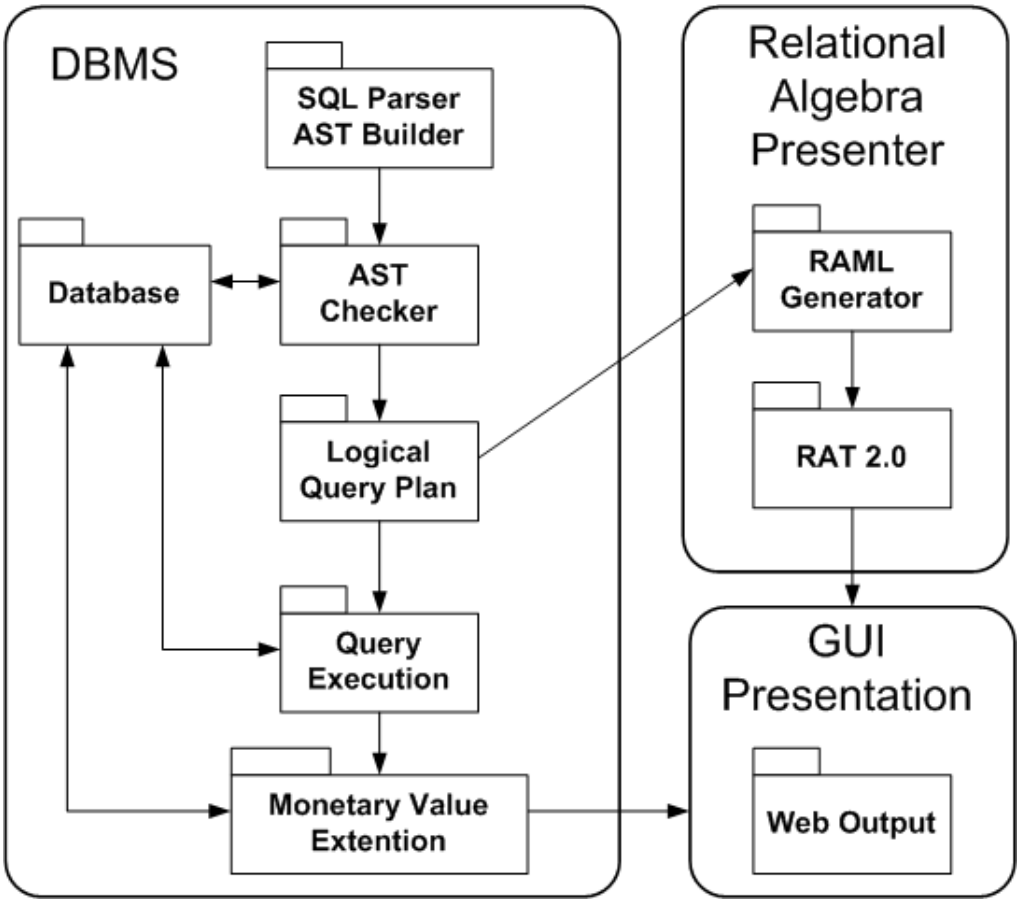

Figure 30: UML package schema of application

In the following, we explain step by step each package and describe internal implementation and their role in the application.

5.2 Database Implementations

Package Database Figure 31 is responsible for storing data of the tables. It contains definition of two object.

- **Table class** - represents class for storing data for the table of database. Every table object has additional variable **mVal**, responsible for storing monetary value of attributes of the table.

- **Database class** - represents additional abstraction layer over tables and aggregates table objects inside. Any access to tables goes through it. Also the global monetary value parameters are defined there **suspicious**, **truncate thresholds** and **reset interval** of accumulated monetary value.

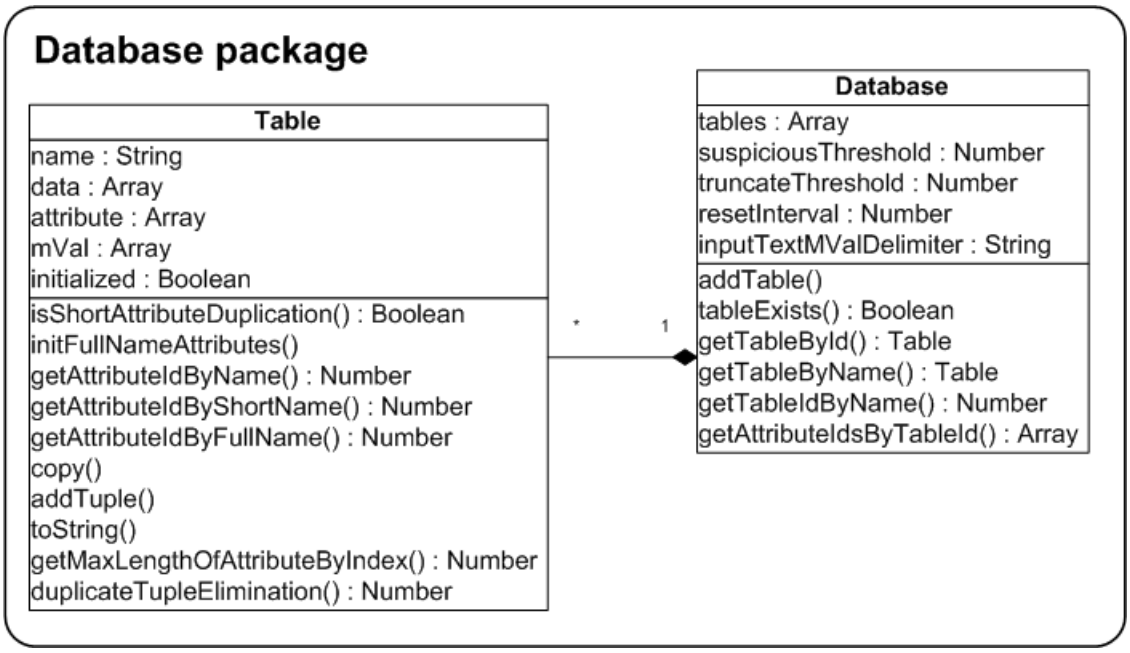

Figure 31: Database UML class diagram

Demonstrator does not support Data Definition Language (DDL). Using GUI it is not possible to directly manipulate conceptual and internal schema of database, any modification to them is possible via changing JavaScript source code. However, the support of DDL was not also defined during the design phase, we consider it for the future work.

Traditional DBMS engines support dozens of different types for the column in a database table, not only for text and numbers, but also for multimedia data like images and video. In our work we limit the list of supported data types to the existing data type of JavaScript language. As extended list of types would only shift attention from demonstration of MVAL approach to the irrelevant tasks. The Table 4 presents supported data types of the our DBMS engine.

[1]Similar to Char, Varchar data type in traditional DBMS
[2]Similar Int, Float, Real data types in traditional DBMS

Data Type	Description
String[1]	represents a sequence of characters
Number[2]	represents integer and floating-point values
Boolean	two possible values: true and false
Null	when attribute is not defined

Table 4: Demonstrator supported data types

Internal schema and data of database is defined during the lunch of the application and any modification of them requires to relaunch the whole demonstrator. On the opposite side, monetary value parameters are changeable in the demonstrator during run time. Listing 7 presents JavaScript code responsible for creating ”Student” table, filling it with data and setting monetary value of every attribute of table.

```
1 var table1_attribute = Array("sid", "name", "address", "major");
2 var table1_data = new Array(Array(101, 'Nathan', 'Edinburg', 'CS'),
3                         Array(105, 'Kolja', 'Edinburg', 'SIM'),
4                         ...
5                         );
6 var table1_mval = new Array (0, 1, 2, 1);
7 var table1 = new Table("Student", table1_attribute,
8         table1_data, table1_mval);
```

Listing 7: Definition of Student table object

```
1  CREATE TABLE Student (
2    sid INT PRIMARY KEY,
3    name VARCHAR(20) NOT NULL MVAL 1,
4    address VARCHAR(20) NOT NULL MVAL 2,
5    major CHAR(2) MVAL 1
6  );
7
8  INSERT INTO Student (sid, name, address, major)
9         VALUES (101, 'Nathan', 'Edinburg', 'CS');
10 INSERT INTO Student (sid, name, address, major)
11        VALUES (105, 'Kolja', 'Edinburg', 'SIM');
12 ...
```

Listing 8: Definition of Student table using traditional DDL

For the comparison our implementation, we presented on the Listing 8 how the same

result could be achieved using DDL in traditional DBMS with support of monetary value preserving concept.

5.3 Query Processing Implementation

SQL query processing, one of the milestone of the work, which allows us not only show some predefined queries with results, but also allows user to interact with demonstrator for better understand the work of MVAL approach.
An overview of query processing schema is presented on the Figure 23, later in this section we explain step by step every part of it.

5.3.1 Query Compiler

Query compiler phase is entry point for all SQL query of a user. Briefly speaking it contains three major steps: abstract syntax tree creation, validation and logical query plan generation. Below we describe each of them in details.

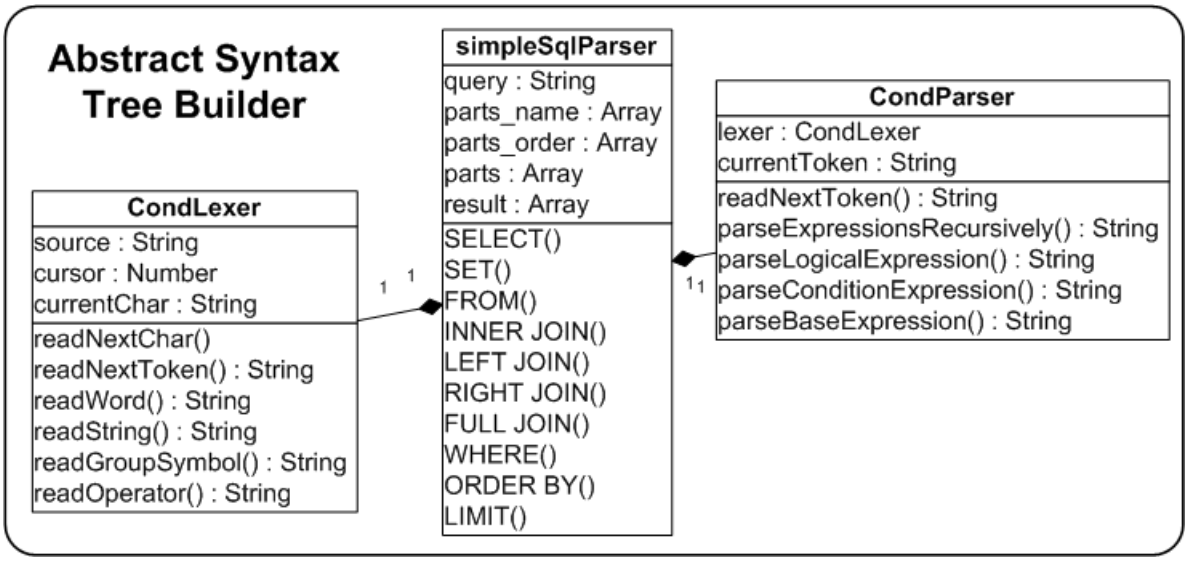

Figure 32: UML class diagram of abstract syntax tree builder package

Abstract Syntax Tree building

We are not the first who stumbles across the problem of converting source code into Abstract Syntax Tree (AST). This problem has been well described in cycle of compiler developers [Mog10], since appearance of first computer languages. For this reason, many open implementations of parser for different languages are exist. The most suitable one, that we could find for our work was simpleSqlParser [Sfe13] library. Its functionality only covers parsing of basic SQL select-from-where statements. As, the simpleSqlParser library is not fully satisfied functional requirements of the work, we have extended [3] it

[3] https://github.com/dsferruzza/simpleSqlParser/commits?author=oshamin

till we cover all requirements, which we described in the Section 4.4.
The library is presented on the Figure 32 and consists of three objects:

- **condLexer** - task of this object is to perform a lexical analysis of the SQL query to transform it from being just a text into a list of ”tokens”.
- **condParser** - responsible for parsing logical and conditional expression of SQL
- **simpleSQLParser** - this object composites two aforementioned objects and recursive goes through SQL text query transforming it into AST.

The Abstract Syntax Tree (AST) is an intermediate representation that captures the relationship between syntactical elements of a computer language in a way that is easier to work with it programmatically. In addition to that, it only keeps the essence structure of the source code and omits the irrelevant details.

For better understanding internal implementation of query processing, we create an example of SQL query on the Listing 9, based on which we make further explanation.

```
1 SELECT * FROM Student
2 JOIN Transcript ON Transcript.sid = Student.sid
3 WHERE Student.sid=101
4   AND Student.major = "CS";
```

Listing 9: SQL query used for demonstration of application internals

Using the SQL query as the input for the AST builder package, we retrieve following AST structure (see Figure 33). Where each node of the tree denotes a construct occurring in the SQL code.

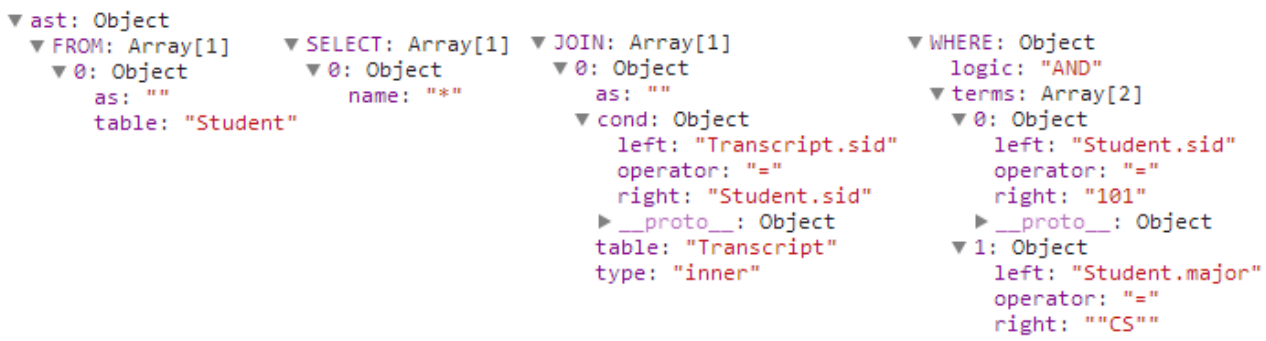

Figure 33: Internal representation of abstract syntax tree of the query

Abstract Syntax Tree Validation

At the step, the built AST for possible semantic, syntax errors is scanned :

- do all tables in FROM and JOIN clauses exist in database?
- do all qualified attributes mentioned in query exist in their tables?
- do all unqualified attributes exist in exactly one referenced table?
- do attribute types correct?

If the any of mentioned errors found, the further execution of processing is interrupted. The error explanation message is contracted and presented to the user.

Logical Query Plan generation

In case, there are no errors in the AST, we build a Logical Query Plan (LQP) out of it. The internal representation of packages is presented on the Figure 34. Logical Query Plan module contains out two objects. One of them **LogicalQueryGraph** is responsible for traversing AST structure and together with underlying object **LogicalQueryNode** build Logical Query Plan (LQP). Other three enumeration types are used as a predefined list with codes of relational algebra operations, relations and conditions inside the every nodes.

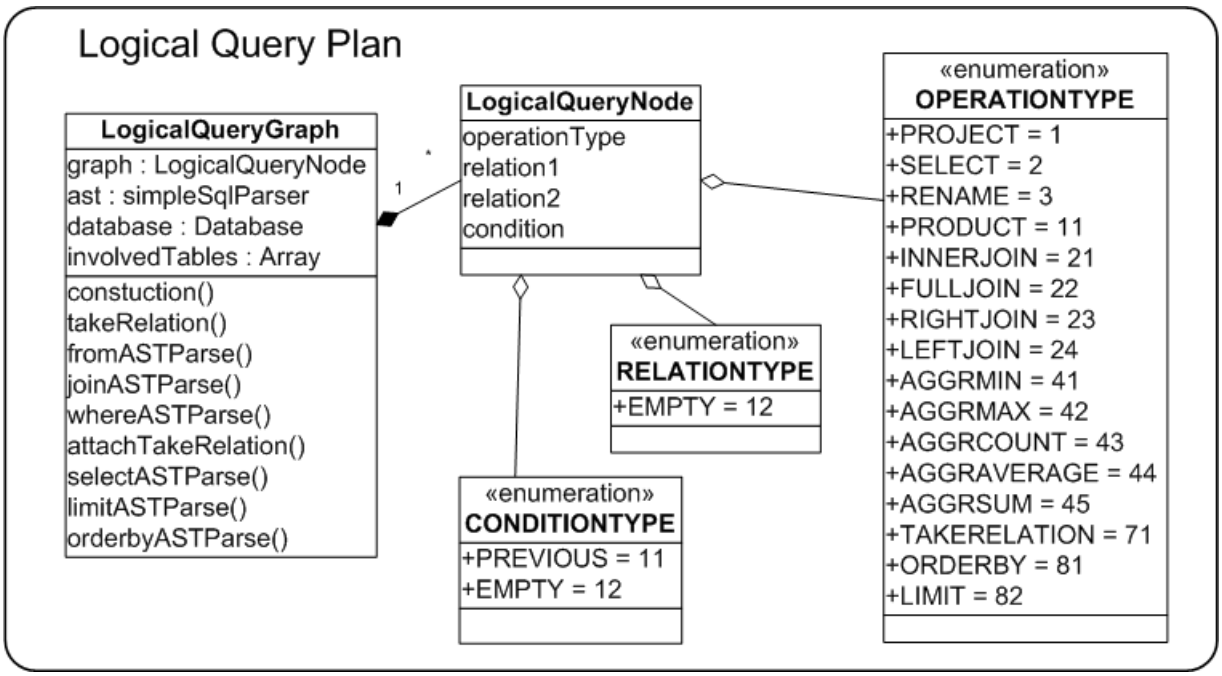

Figure 34: UML class diagram of Logical Query Plan package

Logical Query Plan (LQP) is created as a tree data structure that corresponds to a query graph. Its leaf nodes represent the input relations of the SQL query and internal nodes represent the relational algebra operations, illustrated on the Listing 9.

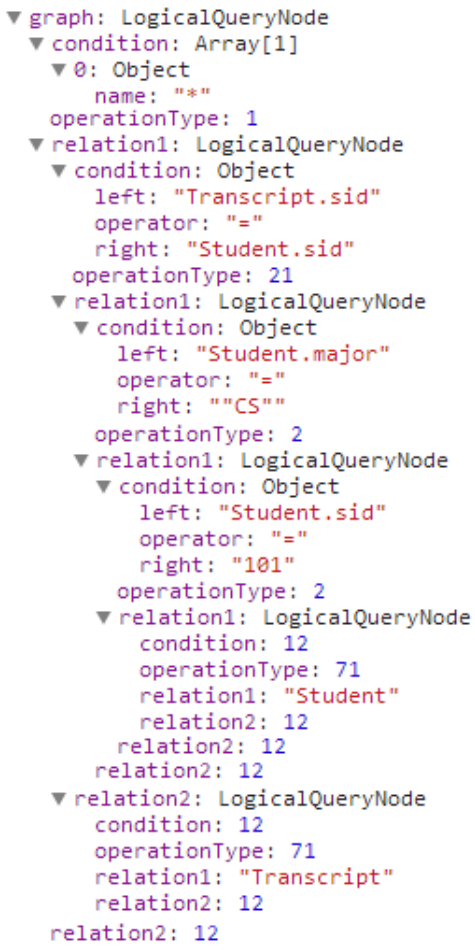

Figure 35: Internal representation of logical query plan

We do not have separate module responsible for logical query optimization. As was written in the requirement Section 4.4, we did not plan to implement complicated optimization steps, because they are out of scope the main aim of the demonstrator. We simplified the optimization part and applied only heuristic rules during step of LQP generation.

5.3.2 Query Execution

At this stage prepared and optimized LQP goes to the module responsible for its sequential execution. An execution of the query consists of sequential executing of internal node operation and then replacing internal node by the resulting relation from executing the operation. The order of execution starts at the leaf nodes, which represent the

input database relations of the query and ends at the root node which represents the query final operation. After the execution of root node is finished, we receive resulting relation for the query.

Due to the one of the application goals, demonstration of data leakage preserving by MVAL approach. We decided to use small-scale tables with straightforward relation schema (see in the Section 4.5), that user can easily catch the main idea. For that reason, implementing complex algorithms for relational algebra operations do not pay off. Due to the fact that the tables are already stored in the memory of client side, by the nature of executing of web application (see in the Section 4.1), and size of tables is less then hundreds kilobytes. Even with straight forward brute force algorithm the demonstrator can produce results less than in a second.

Further, we describe pseudo code of algorithms those we used for relational operation in the project. For the details of their JavaScript implementation we refer reader to the source code.

Algorithm 1 Select algorithm, linear search

```
1: Create new temporary relation T;
2: for each tuple t in input relation R do
3:    evaluate condition for tuple t;
4:    if true then
5:       insert t into T;
6:    end if
7: end for
8: return T;
```

Pseudo code of selection algorithm is presented on the listing of Algorithm 1 with complexity $O(n)$. We used linear search to sequentially check whether attribute values satisfy the selection condition. As the condition could be a very complex, contains combination of logical operation (OR, AND) and brackets of various depth, the internal realization of code line 3 of Algorithm 1 is realized in a recursion.

Algorithm 2 Project algorithm

```
Create new temporary relation T;
for each tuple t = (A_i : d_1, ..., A_n : d_n) in input relation R do
   compute u = (A_i1 : d_i1, ..., A_ik : d_ik); {[i...k] ∈ requested attributes}
   insert u into T;
end for
duplicate elimination in T;
return T;
```

Implementation of a project operation over the relation is presented on the listing of Algorithm 2. The result of the operation contains the same number of tuples as original relation, but with the values for the requested attributes in each tuple. At the end of algorithm we do duplicate tuples elimination using sorting approach in case of tuple repetition in the resulting relation. The idea behind sorting approach is to sort the tuples of the relation using all the remaining attributes as the sort key. This has the effect of arranging the tuples in a way, that duplicates are grouped and can be removed easily afterwards. The resulting complexity of project Algorithm 2 is $O(n * log_2(n))$.

Algorithm 3 Cartesian product algorithm

```
Create new temporary relation T;
for each tuple t in input relation R do
  for each tuple u in input relation S do
    insert t combined with u into T;
  end for
end for
return T;
```

Cartesian product algorithm is presented on the listing of Algorithm 3, it is very expensive operation with complexity $O(n^2)$, as its result includes a record for each combination of records from R and S relations.

The union and intersection algorithms are presented on the listing of Algorithm 4 and Algorithm 5 accordingly. Both algorithms are only applied to relations that has equal number of identical attributes, other words they must be union-compatible. These operations are mainly used for disjunctive conditions, where simple conditions are connected by the logical OR, and conjunctive conditions, where simple conditions are connected by the logical AND. As the SQL parser does not support nested queries (details about them in the Section 2.3.3), it is not possible to embed SQL query within each other.

Algorithm 4 Union algorithm

```
Create new temporary relation T;
for each tuple t in input relation R do
  insert t into T;
end for
for each tuple u in input relation S do
  insert u into T;
end for
duplicate elimination in T;
return T;
```

For the implementation of JOIN operations we used nested-loop join Algorithm 6, with

Algorithm 5 Intersection algorithm

```
Create new temporary relation T;
for each tuple t in input relation R do
  for each tuple u in input relation S do
    if tuple t ≡ table u then
      insert t into T;
    end if
  end for
end for
return T;
```

time complexity $O(n * m)$. It is naive algorithm that joins two set using two nested loops. We chose it for simplification, as its performance does not influence much on total processing time of query. As the total size of database is small and it is located in memory.

Algorithm 6 Join algorithm

```
Create new temporary relation T;
for each tuple t in input relation R do
  for each tuple u in input relation S do
    if join condition is true then
      insert t combined with u into T;
    end if
  end for
end for
duplicate elimination in T;
return T;
```

Algorithms of LEFT OUTER JOIN, RIGHT OUTER JOIN and FULL OUTER JOIN are used the same principle as Algorithm 6. The only difference is how the tuples is combined in case the join condition is false. For example in LEFT JOIN, the tuples with unmatched condition from the left side are initialized with NULL's.

All these operations are executed in a sequence, the output from one operation produces result relation that becomes the input for the subsequent operation.

5.4 Data Leakage Preserving by MVAL Extension

The monetary value preserving module comes to play after the result of query is calculated, in spite of in the paper [BS13a], they recommended to calculation the accumulated monetary value of a query before physically querying the result set. They did this due to remove unnecessary calculation overhead that eventually increase performance

in production DBMS. In our case, we do not have such problem because of the size of tables are quite small. And all calculations are associated with the processing of the request finish within milliseconds. From point of end client, who use the demonstrator, both methods will produce the same results.
The internals of monetary value package is presented on the Figure 36. It consists of three objects:

BucketMonetaryValue - represents ”token”, smallest elements, where we store information about time and monetary cost of executed query.

SlidingWindowMValBucket - represents ”bucket”, where we store all monetary values ”tokens” with time of operation and its monetary value cost.

MonetaryValue - is a manager object, any operation to aforementioned object goes thought it. It determines when monetary value goes over the thresholds and takes action to truncate the output result set.

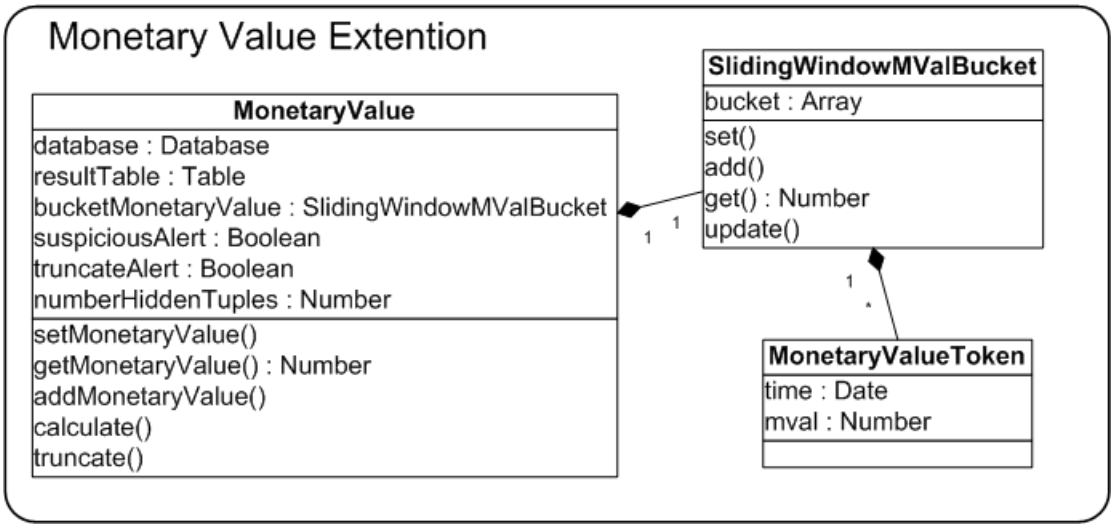

Figure 36: Internal representation of monetary value package

During design step, we decided to implement time sliding method (see details in the Section 4.4.3) for calculation of an accumulated monetary value, as it is more restrictive in comparison with time reset methods. Another argument for this method, that it is also more visually attractive, as the user can see how the accumulated monetary value is rising and decreasing while working with demonstrator instead of each defined time interval dropping an accumulated monetary value to zero. Because of this, we make recalculation of ”bucket” monetary value every second and remove expired ”tokens” to present actual accumulated monetary value of a user.

5.5 Graphical User Interface

While implementing GUI, we followed "Eight Golden Rules of Interface Design" formulated by Shneiderman that we describe in the design Section 4.3 chapter. These rules focus on increasing user's productivity by providing straightforward and universal ways of working with data, comprehensible and rapid informative feedback to enhance the overall perception of the application. They derived heuristically from experience not only him but also other human computer interaction specialist and applicable in most interactive systems. Besides them, we also provided quick access to the most important part of demonstrator.

Those elements of the interface that are responsible for the work of monetary value preserving concept are highlighted relative to all other elements in a light green color. In the Figure 37, we present the GUI of the application, it consists of the following parts:

1. **Part 1** - contains edit box, where user writes SQL query for execution. In case the error appears while processing, the application generates explanation message (what is happened) and present it to the user.
2. **Part 2** - contains global parameters of monetary value violation control method, they are in details explained in Section 3.3. They are all editable.
3. **Part 3** - represent the logging system of traditional DBMS where all exceeding of thresholds are stored for tracing them over time.
4. **Part 4** - represent the relations of the used database, on the top of every relation, user is able to change predefined monetary value for every attribute.
5. **Part 5** - contains of tree tabs, first one is for the output of the resulting relation after checking it on the monetary value violation. Second and third tabs are dedicated for query plan and query relation expression of the executing Logical Query Plan.

Next, we explain how the internal Logical Query Plan (LQP) is transformed into readable for the human query graph and algebra relational expression as well as their rendering on the web page.

5.5.1 Relation Algebra Presentation

While designing the application, one of the requirements was to allow the end user to look under the hood of the query processing and provide the user a visual representation of executing query plan. To realize this opportunity, we need to transform the internal structure of query plan into understandable to the user structure comprising symbols

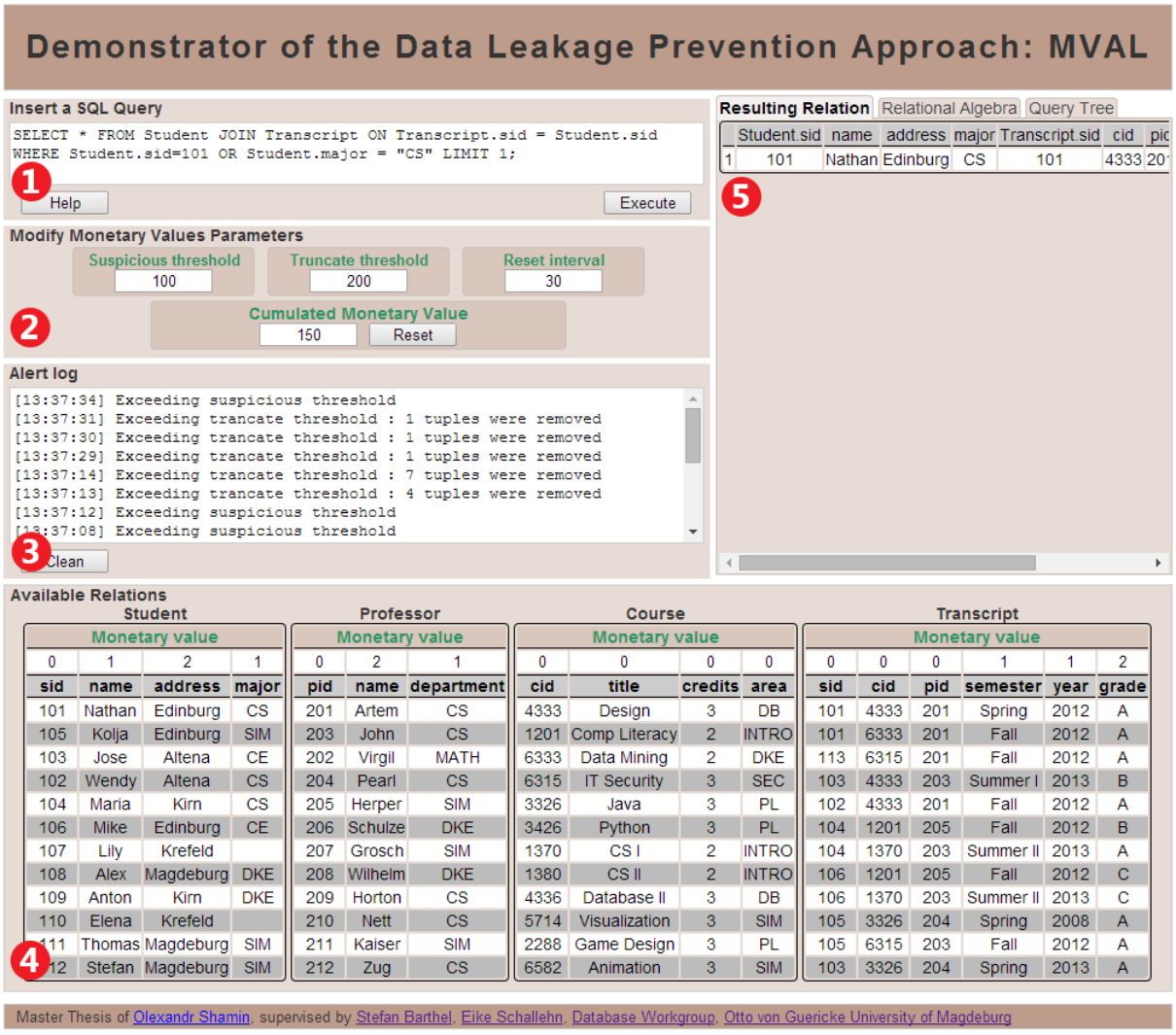

Figure 37: Graphical user interface of the demonstrator

of relational algebra in the form of an expressions and a query tree.

At the beginning we thought to solve this problem by directly encoding relational algebra expressions in HTML page using a Unicode font that supports relational algebra symbols. But we found a project of Texas-Pan American University, that has already faced the same problem what we did. The name of it is Relational Algebra Toolkit 2.0 [AC13]. It is JavaScript library that allows express relational algebra using only web browser, the detailed explanation of it is given in the Section 2.5.3. This library provides many modules (Render, Visualize, Reorder, Translate, Execute, Data Render, Validate), but we are only interested in two of them. One is called RAML Render, it allows to draw relational algebra expressions as part of the text content on the web page. And another one is RAML Visualize, it allows to represent complicated queries as an expression query tree, where the syntactical breakdown is explicitly presented to the viewer with much less visual crowding of information.

Both modules work similarly, they require as input XML document (see details in the Section 2.5.6) constructed using Relational Algebra Markup Language (RAML). It is XML vocabulary for describing relational algebra expressions inside the RAT 2.0 library. As the output they produce serialized XHTML and **S**calable **V**ector **G**raphics elements correspondingly. The limitation of library was in disability to change output elements dynamically, without reloading the entire web page. Because of this we rewrote the source code for needs of project and solved this problem.

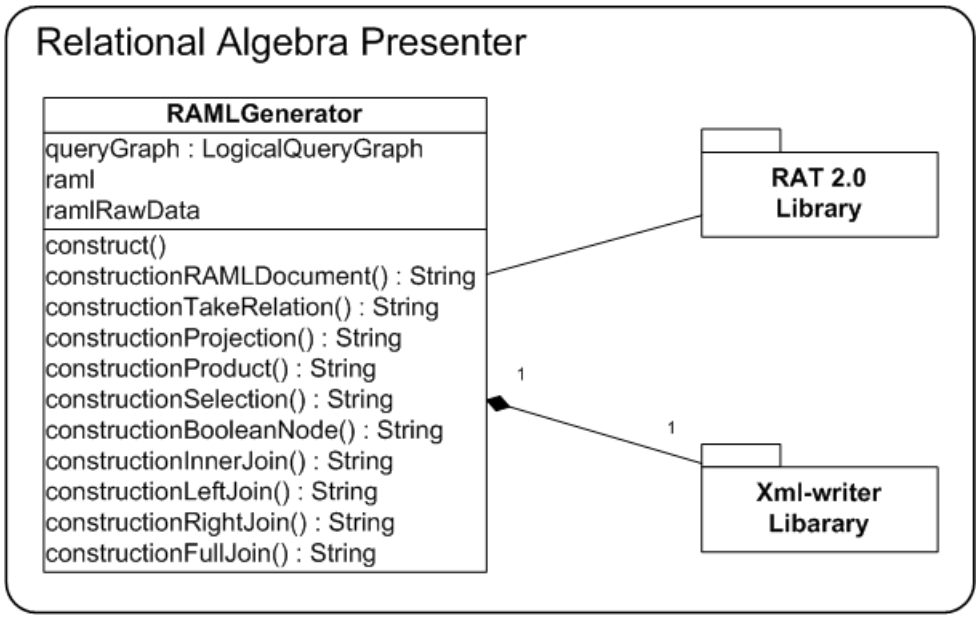

Figure 38: Relational algebra presenter package

Further, we explained how RAT 2.0 library was integrated into our project and present

some example of converting relation algebra into the Relational Algebra Markup Language.
After the logical query plan of initial SQL query is ready and optimized (see detail in the Section 5.3.1), further code executing is divided. One part of code is continue query processing, step by step executing instruction of relation algebra to retrieve final relation. Another part of code starts converting logical query plan into understandable to RAT 2.0 library XML document. It took us to use external library for in-memory XML creation to speed up this process as the native JavaScript functionality for working with XML is quit tedious. The name of library is called xml-writer [Tho14]. XML-writer library represents a writer that provides a non-cached, forward-only means of generating streams containing XML data.

The internal implementation of relational algebra presenter package is presented on the Figure 38. RAMLGenerator object takes as input the Logical Query Plan of the initiation SQL query, traverse it recursively from bottom up. And out of nodes of LQP it contracts the XML document using the xml-writer library according to Relational Algebra Markup Language (RAML) vocabulary. Afterwards this XML document is sent as the input to RAML Render and RAML Visualize modules of RAT 2.0 library for rendering them on the web page in form readable relational algebra expression and query graph.

As an example how to contract RAML document, we illustrate both relation algebra expression and their representation in RAML vocabulary. The RAML document for the select operation (Equation 5.1) presented below on the Listing 10.

$$\sigma_{year>2000}(R) \tag{5.1}$$

```
1  <?xml version="1.0"?>
2  <raml xmlns="http://www.cs.panam.edu/2010/RAML">
3      <expression>
4          <selection/>
5          <boolean>
6              <gt/>
7              <attribute><name>year</name></attribute>
8              <number>2000</number>
9          </boolean>
10         <expression>
11             <relation><name>R</name></relation>
12         </expression>
13     </expression>
14 </raml>
```

Listing 10: XML document based on relational algebra markup language vocabulary

Combining RAML expressions inside each other, we can interpret relation algebra expression of any complexity in the XML form.

At this step, we finished describing components from which the demonstrator application consists, such as Graphical User Interface, query processing and data leakage preserving by MVAL extension. And we presented how they were integrated with each other. Next, the final conclusion over the work is given.

6. Conclusion and Future Work

In this part, we summarize the result of the thesis. At first, the briefly recapitulation of motivation and objectives of the work. Afterwards, we evaluate our work, going through task by task and answering on the following question "How broadly the initial task was solved?". At the end, the future work will be described.

According to the latest security reports, the highest rate among all business assets by insider misuse are databases breaches. As they are the heart of organization and store financial, customer and similar confidential business data. Major traditional mechanisms for protection data are focused on protection against unauthorized access, that allows trusted user (employee, contractor or trusted business partner) intentionally used granted access for commit insider attacks. Every such insider incident has different motivations, from coping sensitive customer data for the further obtaining financial goal by selling it on a black market, to taking scientific research or client database for starting own business or taking higher job position in a competitor organization. Moreover, these incidents can be grouped by one important factor, that sensitive data electronically leaves the organization boarders.

To address these problems, thesis was focused on solving aforementioned sub-tasks (Section 1.2). Below, details of every solved task are presented :

Task 1. Overview, comparison and evaluation data leakage countermeasures.

For solving this task, in the chapter Chapter 3 the overview, the existing insider mitigation approaches were presented and compared with MVAL approach. The comparison of selected approaches with MVAL approach was presented in the Table 1. Also, missed points of MVAL approach, such as calculation of accumulated monetary value of the

same repeatedly accesses data and the possibility to perform sensitive data disclosure via non-sensitive data, were explored in the Section 3.3.2.

Task 2. Choose running environment for MVAL demonstrator platform..

In the work, the review of existing software architectures (native / desktop and web) were carried through. After weak points of both architecture were examined such as access, updated, cross-platform and security problems. We chose a hybrid web applications, which omits weak side of pure desktop and web applications and includes only strong side both of them. Based on the type of architecture we briefly examined existing stack of tools for the application development and chose following one (JavaScript, HTML, CSS) as the superior one.

Task 3. Develop core DBMS functionality with support of SQL query language.

For the purpose of studying the data leakage preserving by MVAL approach as well as comprehensive demonstration its work. We developed DBMS engine with support of SQL query language. As the central idea behind data leakage persistent approach is to protect DBMS from intentional data extraction, part of SQL functionality that responsible for modification or deletion functionality were omitted. Another argument for the realization of SQL language was to allows user, who traditionally works with DBMS such as database administrators and programmers, to understand how the demonstrated data leakage preserving concept would work on the production DBMS, where SQL is de-facto language standard.

To achieve this goal, we examined existing implementation of DBMS engine. However, existing open JavaScript implementations realized only simple storing and extraction functionality over a group of records without support of SQL. Their adaptation to the requirements of demonstrator would require from us to spend an enormous amount of time for understanding their internal implementation for the further redesigning. For that reason, we developed JavaScript DBMS engine from scratch.

Schema of traditional SQL query processing was explored and adapted for the requirements of the demonstrator. It is important to note, that query optimization phase was simplified in our engine. As the response time on the SQL request is less than a second there is no need in the deep optimization. Nevertheless, we build heuristic optimization rules to illustrate, that we also thought about this problem.

In the demonstrator was implemented significant part of SQL extraction statements and their modification such as SELECT, WHERE, INNER JOIN, RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, ORDER BY and LIMIT. Besides that, there was implemented support of condition and comparison operations with support of

brackets for grouping them. Using developed SQL functionality, demonstrator allows a user in a similar way to communicate with it as a user does in a production environment.

Task 4. Integrate MVAL approach into developing DBMS engine.

After the deep MVAL approach description in the Section 3.3, it was integrated into developed DBMS engine. Schema of SQL query processing consists of three main phases: Query compilation, Query Optimization and Query Execution. We implemented MVAL approach at the phase of Query Execution, as at this phase the resulting data is constructed and prepared for the output to the user. In case the resulting data violates the rules of data leakage preserving concept, the necessary filtration over them is performed. For that reason, MVAL approach was located as close as possible to the data output.

Task 5. Design and implementation of usable Graphical Interface.

During this step, we researched possible characteristics to which we could strive to create usable graphical interface in the Section 4.3. It was presented, that usability is subdivided into three sub-factor (ISO 9241-11 [ISO98]) : effectiveness, efficiency and satisfaction.
To address these sub factors, we applied to GUI of demonstrator the "Eight Golden Rules of Interface Design" formulated by Shneiderman [SP05], after properly redefining them for our environment. These rules derived heuristically from experience not only him but also other human-computer interaction specialist and polished over two decades. These rules are focused on improving usability of most interactive systems by providing clear data-entry procedures, comprehensible displays, prevention user error, rapid informative feedback and reducing short-term memory load [SP05]. We understand that any list of rules like this can not be complete, but they provide a good starting point for further improvements.

Task 6. Present evaluation of the implemented platform.

As the demonstrator consist of many components: architecture, DBMS engine, data leakage preserving by MVAL approach, graphical user interface. A summary evaluation of the work consists of an evaluation of every step.

We start from the overall architecture of the application. During the phase of design the two fundamentally different type of software architecture were examined. Also, there were defined their weak and strong sides, based on them. We developed hybrid web application that inherited mostly strong sides of both architectures.

The most time during application development phase took an implementation of JavaScript DBMS engine with support of SQL language. As, execution of human written text is

not trivial task, even if it is structured by predefined sort of rule. It is also important to notice, that from the correct functioning DBMS engines depends further correct work of insider mitigation defense, as it is built upon DBMS. Due to limit of time, we could not realize full functionality of traditional DBMS, for that reason we narrow down functionality to essential for comprehensive demonstration of data leakage preserving approach. For that reason, only one heuristic optimization rule was implemented (push down selection), other two were omitted (push down projection and push down joins) and left for the future work over the application. Nevertheless, all other requirements that were defined in the beginning the development, were solved.

Other vital part of application development was integration of data leakage preserving by MVAL approach with DBMS engine. During this phase, we extended the MVAL approach in the demonstrator application. We offered additional method how to treat reset interval for accumulation monetary value of resulting data and compared original method of Barthel (fixed time interval) with the new one (sliding fix time interval) in the Section 4.4.3. For the reasons of better visual demonstration of MVAL approach, in the application the version with sliding fix time interval was implemented. Also, there were found missed points of MVAL, which we covered in the Section 3.3.2.

The graphical interface of the application was developed to clearly exhibits the main properties of the data leakage prevention by MVAL approach and was targeted to a user, who traditionally closely works with DBMS such as database administrators and programmers. Besides that, in the application the list of rules which increase overall usability was implemented. First, the consistent minimalistic design with focus of demonstration underlying data leakageMVAL concept was build. Second, the internal DBMS functionality allows us to reach immediate feedback on any user action and creates internal locus of control over the application. Third, as the GUI is structured on the five functional blocks: SQL data entry block, monetary value parameters of data leakage concept block, alert log, block with available relation and block with results that focuses user attention on small action groups. Fourth, in case of incorrect input of SQL query, the application presents explanation of an error, that helps the user to recover from it. Fifth, the major actions are reversible, that allows user to return to the initial state. Sixth, necessary information for interaction is presented within one page, that does not create any load for short-memory of the user. The only question that was partially opened is universal usability. To achieve it, the application must be adopted for different user types (examples: novice, knowledgeable, expert) and devices types (examples: size of screen, input interface) that leads us to shift attention from main goal of data leakage preserving by MVAL demonstration to irrelevant tasks. For that reason, we made recommendation on the executing hardware and target user group of application in the Section 4.3.

We want to notice, that during development phase there were not any cost spent, as we used free software products. The working version of the application was deployed

in the web and accessible by the following address : http://mval.bartoldi.de.

Summarizing what was described above, all sub-tasks were entirely solved. Recommended future work is presented below.

6.1 Future Work

Several topics remain open for the future work :

1. During the GUI design phase, we implemented the list of rule for increasing usability.

For comprehensive usability evaluation survey must be carried out, where user accomplish predetermined goals and the measures of accuracy and completeness, as well as its time effort, are recorded. Based on received result the numerical evaluation of usability is possible.

2. During the limit of time for the thesis, not full functionality of traditional DBMS was implemented. Here, we suggest to extend it by the following functions :

- added support for DDL statements (RENAME, CREATE, DROP TABLE, etc.)
- added support for nested queries.
- added support for aggregation operation with and GROUP BY statement.
- added support for scalar and user-defined functions.

Offered functionality must closer simulate traditional DBMS, that helps a user to deeply understand further improvements of MVAL approach.

Bibliography

[AC13] Hussein Bakka Artem Chebotko, Nathan Arnold. Relational Algebra Toolkit 2.0. Website, October 2013. Available online at http://rat.cs.panam.edu/RAT2/; visited on Jun 6th, 2014. (cited on Page xi, 26, and 74)

[Ark13] Brad Arkin. Important Customer Security Announcement. Website, October 2013. Available online at http://blogs.adobe.com/conversations/2013/10/important-customer-security-announcement.html; visited on May 21th, 2014. (cited on Page 1)

[Bel05] David Elliott Bell. Looking back at the bell-la padula model. In *ACSAC*, volume 5, pages 337–351, 2005. (cited on Page 21)

[Bib77] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, DTIC Document, 1977. (cited on Page 23)

[Bin92] Leonard J Binns. Inference through secondary path analysis. In *Sixth Working Conference on DATABASE SECURITY*, page 203. DTIC Document, 1992. (cited on Page 24)

[BN89] David FC Brewer and Michael J Nash. The chinese wall security policy. In *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*, pages 206–214. IEEE, 1989. (cited on Page 23)

[BS13a] Stefan Barthel and Eike Schallehn. The monetary value of information: A leakage-resistant data valuation. In *BTW Workshops*, pages 131–138, 2013. (cited on Page iii, xv, 3, 29, 36, 39, 41, 49, 55, and 70)

[BS13b] Stefan Barthel and Eike Schallehn. MVAL: addressing the insider threat by valuation-based query processing. In *Proceedings of the 25th GI-Workshop "Grundlagen von Datenbanken 2013", Ilmenau, Germany, May 28 - 31, 2013*, pages 58–63, 2013. (cited on Page iii, xi, 3, 29, 38, 40, and 41)

[Bül10] Alfred Büllesbach. *Concise European IT Law*. Concise IP. Kluwer Law International, 2010. (cited on Page 2)

[Byr08] Eric Byres. Defense in depth. *Control Engineering Asia June 2008*, 2008. (cited on Page 39)

[CB01] Thomas M. Connolly and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2001. (cited on Page 5)

[CB05] Thomas M Connolly and Carolyn E Begg. *Database systems: a practical approach to design, implementation, and management*, volume 1. Pearson Education, 2005. (cited on Page 5, 16, and 17)

[Cen14a] InfoWatch Analytical Center. Global Data Leakage Report, 2013. Technical report, InfoWatch Analytical Center, 2014. (cited on Page xi, 1, 33, and 34)

[Cen14b] Verizon Analytical Center. 2014 Data Breach Investigations Report. Technical report, Verizon, 2014. (cited on Page 2)

[CGL00] Christina Yip Chung, Michael Gertz, and Karl Levitt. Demids: A misuse detection system for database systems. In *Integrity and Internal Control in Information Systems*, pages 159–178. Springer, 2000. (cited on Page 34 and 35)

[Cha98] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43. ACM, 1998. (cited on Page 17)

[Che76] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976. (cited on Page 12)

[CIS14] Cisco 2014 Annual Security Report. Technical report, 2014. (cited on Page 1)

[CMT12] Dawn M Cappelli, Andrew P Moore, and Randall F Trzeciak. *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Addison-Wesley, 2012. (cited on Page 33)

[Cod70] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. (cited on Page 13)

[Cod82] Edgar F Codd. Relational database: a practical foundation for productivity. *Communications of the ACM*, 25(2):109–117, 1982. (cited on Page 6)

[Cod85] Edgar F Codd. Does your dbms run by the rules? *Computer World*, 21:11, 1985. (cited on Page 13)

[Cro08] Douglas Crockford. The World's Most Misunderstood Programming Language Has Become the World's Most Popular Programming Language. Website, March 2008. Available online at http://javascript.crockford.com/popular.html; visited on May 22th, 2014. (cited on Page 25)

[Den85] Dorothy E. Denning. Commutative filters for reducing inference threats in multilevel database systems. *2013 IEEE Symposium on Security and Privacy*, 0:134, 1985. (cited on Page 24)

[EF14] Inc. Eclipse Foundation. Eclipse - The Eclipse Foundation open source community website. Website, May 2014. Available online at http://www.eclipse.org; visited on May 28th, 2014. (cited on Page 46)

[EN10] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition, 2010. (cited on Page xi, 5, 6, 10, 11, 12, 13, and 18)

[Eri12] Birger Eriksson. The Document Object Model used to access objects in web pages with eg. javascript. Website, January 2012. Available online at http://commons.wikimedia.org/wiki/File:DOM-model.svg; visited on Jun 20th, 2014. (cited on Page xi and 28)

[FJ02] Csilla Farkas and Sushil Jajodia. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter*, 4(2):6–11, 2002. (cited on Page xi and 23)

[FK09] David F Ferraiolo and D Richard Kuhn. Role-based access controls. *arXiv preprint arXiv:0903.2171*, 2009. (cited on Page 22)

[Fla06] David Flanagan. *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2006. (cited on Page 5 and 24)

[Gen12] Stuart Gentry. Access Control: Models and Methods. Website, November 2012. Available online at http://resources.infosecinstitute.com/access-control-models-and-methods/; visited on Jun 13th, 2014. (cited on Page 20)

[GR12] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the Future*, 2012. (cited on Page 31)

[Gra93] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys (CSUR)*, 25(2):73–169, 1993. (cited on Page 17)

[GTD98] Stephen E. Fienberg George T. Duncan. Obtaining information while preserving privacy: A markov perturbation method for tabular data. *Statistical Data Protection*, 1998. (cited on Page 24)

[HCCY13] Carly L Huth, David W Chadwick, William Claycomb, and Ilsun You. Guest editorial: A brief overview of data leakage and insider threats. *Information Systems Frontiers*, 15(1):1–4, 2013. (cited on Page 32)

[Hin88] Thomas H Hinke. Inference aggregation detection in database management systems. In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on*, pages 96–106. IEEE, 1988. (cited on Page 24)

[Hol13] Rich Holowczak. DATABASE MANAGEMENT SYSTEMS COURSE NOTES. Website, January 2013. Available online at http://holowczak.com/database-management-systems-course-notes/; visited on Jun 29th, 2014. (cited on Page xi and 7)

[HSRE12] Amir Harel, Asaf Shabtai, Lior Rokach, and Yuval Elovici. M-score: A misuseability weight measure. *Dependable and Secure Computing, IEEE Transactions on*, 9(3):414–428, 2012. (cited on Page 36)

[INC04] ANSI INCITS. Role based access control. *INCITS 359-2004*, 2004. (cited on Page 22)

[ISO98] ISO. Ergonomic requirements for office work with visual display terminals (vdt)s - part 11 guidance on usability. ISO 9241-11: 1998, International Organization for Standardization, Geneva, Switzerland, 1998. (cited on Page 47 and 79)

[ISO00] ISO. Information technology — software product quality — part 1: Quality model. ISO ISO/IEC FDIS 9126-1:2000, International Organization for Standardization, Geneva, Switzerland, 200. (cited on Page 47)

[Jär11] Hannu Järvinen. Html5 web workers. *T-111.5502 Seminar on Media Technology B P, Final Report*, 2011. (cited on Page 27)

[JN14] Manuel Reinartz Julia Neuhauser. Datenleck: 400.000 vertrauliche Schülertests im Internet aufgetaucht. Website, February 2014. Available online at http://diepresse.com/home/bildung/schule/1567203/Datenleck_400000-vertrauliche-Schulertests-im-Internet-aufgetaucht; visited on May 21th, 2014. (cited on Page 1)

[KTB08] Ashish Kamra, Evimaria Terzi, and Elisa Bertino. Detecting anomalous access patterns in relational databases. *The VLDB Journal*, 17(5):1063–1077, 2008. (cited on Page 35)

[LJ92] Carl E. Landwehr and Sushil Jajodia, editors. *Database Security, V: Status and Prospects, Results of the IFIP WG 11.3 Workshop on Database Security, Shepherdstown, West Virginia, USA, 4-7 November, 1991*, volume A-6 of *IFIP Transactions*. North-Holland, 1992. (cited on Page 24)

[Lun89] T.F. Lunt. Aggregation and inference: facts and fallacies. In *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*, pages 102–109, May 1989. (cited on Page 24)

[Mar96] Donald G. Marks. Inference in mls database systems. *Knowledge and Data Engineering, IEEE Transactions on*, 8(1):46–55, 1996. (cited on Page 24)

[McC08] Michael McCormick. Data theft: a prototypical insider threat. In *Insider Attack and Cyber Security*, pages 53–68. Springer, 2008. (cited on Page iii and 34)

[Mog10] Torben Ægidius Mogensen. *Basics of Compiler Design*. Torben Ægidius Mogensen, 2010. (cited on Page 5 and 64)

[MPNU10] Sunu Mathew, Michalis Petropoulos, Hung Q Ngo, and Shambhu Upadhyaya. A data-centric approach to insider attack detection in database systems. In *Recent Advances in Intrusion Detection*, pages 382–401. Springer, 2010. (cited on Page 34 and 35)

[NC13] Richard Powell Nick Cappi. Presentation : INTEGRITY FOR CYBER SECURITY. Technical report, Technology Conference 2013, 2013. (cited on Page xi and 18)

[Net12] Microsoft Developer Network. Unleash the Power of Hardware-Accelerated HTML5 Canvas. Website, January 2012. Available online at http://msdn.microsoft.com/en-us/hh562071.aspx; visited on Jun 19th, 2014. (cited on Page 25)

[PG06] Joon S Park and Joseph Giordano. Access control requirements for preventing insider threats. In *Intelligence and Security Informatics*, pages 529–534. Springer, 2006. (cited on Page 31)

[PN09] Peter Wahl Paul Needham. Oracle label security in government and defense environments. *An Oracle White Paper*, 2009. (cited on Page 23)

[RB14] Travis Leithead Robin Berjon, Steve Faulkner. HTML5 A vocabulary and associated APIs for HTML and XHTML. Website, June 2014. Available online at http://www.w3.org/TR/html5/; visited on Jun 20th, 2014. (cited on Page 27)

[RGG03] Raghu Ramakrishnan, Johannes Gehrke, and Johannes Gehrke. *Database management systems*, volume 3. McGraw-Hill New York, 2003. (cited on Page 5 and 13)

[SCM+12] George Silowash, Dawn Cappelli, Andrew Moore, Randall Trzeciak, Timothy J Shimeall, and Lori Flynn. Common sense guide to mitigating insider threats 4th edition. Technical report, DTIC Document, 2012. (cited on Page 32)

[Sfe13] David Sferruzza. Javascript library to parse CRUD (Create Retrieve Update Delete) SQL queries. Website, January 2013. Available online at https://github.com/dsferruzza/simpleSqlParser; visited on Jun 4th, 2014. (cited on Page 64)

[SFK00] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The nist model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 2000, 2000. (cited on Page 22)

[SP05] Shneiderman Ben Shneiderman and Catherine Plaisant. Designing the user interface 4 th edition. *ed: Pearson Addison Wesley, USA*, 2005. (cited on Page 47 and 79)

[SPC+10] Neeraj Sharma, Liviu Perniu, Raul F Chong, Abhishek Iyer, Chaitali Nandan, Adi-Cristina Mitea, Mallarswami Nonvinkere, and Mirela Danubianu. database fundamentals. *Publisher: IBM Corporation*, 2010. (cited on Page xi and 12)

[Sql10] SqlPac. Three-valued logic (3VL) truth table. Website, March 2010. Available online at http://en.wikipedia.org/wiki/File:3VL_Truth_Table.png; visited on Jun 20th, 2014. (cited on Page xi and 15)

[Sta14] StatCounter Global Stats. Compatibility table for support of SVG in desktop and mobile browsers. Website, May 2014. Available online at http://caniuse.com/svg; visited on Jun 20th, 2014. (cited on Page 29)

[Sur14a] W3Techs Web Technology Surveys. Usage of client-side programming languages for websites. Website, January 2014. Available online at http://w3techs.com/technologies/overview/client_side_language/all; visited on Jun 26th, 2014. (cited on Page 46)

[Sur14b] W3Techs Web Technology Surveys. Usage of JavaScript libraries for websites. Website, January 2014. Available online at http://w3techs.com/technologies/overview/javascript_library/all; visited on Jun 19th, 2014. (cited on Page 25)

[SVEG10] Erez Shmueli, Ronen Vaisenberg, Yuval Elovici, and Chanan Glezer. Database encryption: an overview of contemporary challenges and design considerations. *ACM SIGMOD Record*, 38(3):29–34, 2010. (cited on Page 40)

[Tho14] Nicolas Thouvenin. xml-writer - Native and full Javascript implementation of the classic XMLWriter class. Website, May 2014. Available online at https://github.com/touv/node-xml-writer; visited on Jun 9th, 2014. (cited on Page 75)

[TWH05] Paul Meijer Tim Wagner, Ted Bashor and Pieter Humphrey. Overview of the Eclipse Web Tools Platform. Website, September 2005. Available online at http://www.oracle.com/technetwork/articles/entarch/eclipse-web-tools-platform-093378.html; visited on May 28th, 2014. (cited on Page 46)

[Uni13] Carnegie Mellon University. 2013 US State of Cybercrime Survey, How Bad is the Insider Threat? Technical report, Carnegie Mellon University, 2013. (cited on Page xi, 1, 32, and 33)

[Wil88] Jackson Wilson. Views as the security objects in a multilevel secure relational database management system. In *IEEE Symposium on Security and Privacy*, pages 70–84, 1988. (cited on Page 40)

[YL98] Raymond W Yip and Karl N Levitt. Data level inference detection in database systems. In *Computer Security Foundations Workshop, 1998. Proceedings. 11th IEEE*, pages 179–189. IEEE, 1998. (cited on Page 24)

[Yug06] Yug. Demonstration of differences between bitmapped and SVG images. Website, October 2006. Available online at http://commons.wikimedia.org/wiki/File:Bitmap_VS_SVG.svg; visited on Jun 20th, 2014. (cited on Page xi and 29)

[ZZ04] Jensen J Zhao and Sherry Y Zhao. Internet technologies used by inc. 500 corporate web sites. *Issues in Information Systems*, 4(20):04, 2004. (cited on Page 46)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den July 11, 2014