ANOMALY DETECTION TECHNIQUES FOR THE PROTECTION OF DATABASE SYSTEMS AGAINST INSIDER THREATS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Asmaa M. Sallam

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

To Medhat
For raising me up

TABLE OF CONTENTS

Page

LIST OF TABLES vii
LIST OF FIGURES viii
ABBREVIATIONS x
ABSTRACT xi
1 INTRODUCTION 1
2 STATE OF THE ART 5
2.1 Scope and Methodologies 5
2.1.1 Features Space 5
2.1.2 Methodologies and Sources of Queries 7
2.2 Prominent Techniques and Systems 8
2.2.1 Syntax-based AD 8
2.2.2 Data-centric AD 9
2.2.3 Temporal AD 10
2.2.4 Profiling Application Programs 13
3 DETECTION OF ANOMALOUS QUERIES THAT RETRIEVE DATASETS WHOSE SIZES EXCEED THE NORMAL SIZES 18
3.1 Architecture that Directly Supports AD in Relational Databases 19
3.2 Data Representation 20
3.3 Extraction of Selectivity Information 22
3.3.1 Algorithms 22
3.3.2 T-DBMS Adapters 27
A. Oracle T-DBMS Adapter 27
B. SQL Server T-DBMS Adapter 29
3.4 Role-based AD 29
3.4.1 The Binary Classifier 29
3.4.2 The Naive Bayesian Classifier (NBC) 30
3.4.3 The Multi-Labeling Classifier (MLC) 33
3.5 User-based AD 34
3.6 Taxonomy of Anomalies 35
3.7 Experimental Evaluation 37
3.7.1 Set A – Experiments in Collaboration with NG 37
3.7.2 Set B – Experiments using Synthetic Datasets 42
A. Test Scenarios 42

Page
B. Results for Role-based AD 44
C. Results for User-based AD 45
3.7.3 Set C – Experiments using the OLTP-Benchmark 46
3.8 Conclusions 47

4 MONITORING THE FREQUENCIES OF EXECUTION OF PERIODIC QUERIES 54
4.1 Architecture and Scope of the work 55
4.1.1 Use Case 55
4.1.2 Architecture 56
4.2 Data Representation 58
4.2.1 SQL Queries Signatures 58
4.2.2 Time Series Representation 62
4.3 Training Phase 62
4.3.1 Periodicity Detection 63
4.3.2 Eliminating False-Positives 65
4.3.3 Inferring the Relationships between Periodic Queries 67
4.3.4 Format of Profiles 68
4.4 Detection Phase 69
4.5 Experimental Evaluation 70
4.6 Conclusions 72

5 DETECTION OF ANOMALOUS SEQUENCES OF QUERIES THAT RETRIEVE DATASETS LARGER THAN NORMAL SIZES 73
5.1 Architecture 74
5.2 Session Tracking 76
5.2.1 Detection of Anomalous Queries 76
5.2.2 Detection of Anomalous Query Sequences 78
5.3 Session Evaluation 82
5.3.1 Basic Approach 82
A. Session Representation 83
B. Algorithms 83
5.3.2 Session Evaluation with Lengths' Partitioning 84
5.4 Experimental Evaluation 85
5.4.1 Results of the Evaluation of ST 86
A. False-Positives 87
B. False-Negatives 87
5.4.2 Results of the Evaluation of SE 88
A. False-Positives 88
B. False-Negatives 89
5.4.3 Results of the Evaluation of SE+ 91
5.4.4 Concluding Remarks 92
5.5 Conclusions 93

Page

6 DETECTION OF ANOMALIES IN RATES OF TABLES REFERENCES AND TUPLES RETRIEVALS . . . 94
6.1 Architecture . . . 95
6.2 Training Phase . . . 98
6.2.1 Organizing Logs . . . 100
6.2.2 Building Profiles . . . 100
6.3 Detection Phase . . . 105
6.4 Experimental Evaluation . . . 109
6.4.1 False-Positives . . . 109
6.4.2 False-Negatives . . . 111
6.4.3 Concluding Remarks . . . 115
6.5 Conclusions . . . 117

7 FUTURE WORK . . . 118

REFERENCES . . . 122

LIST OF TABLES

Table Page

3.1 Quadruplet representation of queries in role-based and user-based AD . . . 23

3.2 Description of training data used in the evaluation of DBSAFE 39

3.3 First run of evaluation of DBSAFE 41

3.4 Second run of evaluation of DBSAFE 41

3.5 Training time (in mins) of role-based and user-based AD 48

3.6 Results of the evaluation of the NBC used in role-based AD 52

3.7 Results of the evaluation of the MLC used in role-based AD 52

3.8 Results of the evaluation of the COBWEB used in user-based AD 53

5.1 Example DB and sessions' logs 78

5.2 Example ST profile 79

6.1 Data access rates 99

6.2 Example DB table 99

6.3 Example roles and users sub-logs 99

6.4 Maintenance of time-series by result-based AD ($L_{\text{res}} = 20$ mins, $L_{\text{i}} = \{2, 3\}, L_{\text{s}} = 3$) 100

6.5 Configuration parameters of result-based AD 104

6.6 Result-sets of an example query and its modification after adding the primary-keys of the range-tables 107

6.7 Detection phase data-structures corresponding to one DB table 108

LIST OF FIGURES

Figure Page

3.1 DBSAFE Architecture 21

3.2 Two possible optimizer plans for an example query 26

3.3 Optimizer's plan for a query containing an equi-join operator 26

3.4 DBSAFE evaluation environment 40

3.5 Test case used for the evaluation of role-based and user-based AD 48

3.6 Results of the evaluation of role-based AD 49

3.6 Results of the evaluation of role-based AD 50

3.7 Results of the evaluation of user-based AD 51

4.1 Representation of SQL queries for the purpose of the detection of periodic queries 59

4.2 Time series representation of query execution time-stamps 63

4.3 Result of computing the autocorrelation for a periodic time series with period $p = 5$ 66

4.4 Algorithm Period-Filter search example ($w = p = 3$, $i = 1$) 66

4.5 Example PAG 67

5.1 Timing of the application of the AD techniques used for the detection of temporal data ex-filtration 75

5.2 Results of the evaluation of ST 86

5.3 Accuracy of the detection of anomalous outliers by SE 91

5.4 Accuracy of the detection of anomalous inliers by SE 92

6.1 Steps for the inspection of the result-set rows of a new user query (Q) . . . 96

6.2 Data-structures used during the detection phase of result-based AD . . . 107

6.3 False-positive errors by preliminary inspection 112

6.4 False-positive errors by deep inspection 113

6.5 False-negative errors by preliminary inspection 115

6.6 False-negative errors by deep inspection 116

ABBREVIATIONS

AD	Anomaly detection
DB	Database
DBMS	Database management system
RBAC	Role-based access control
QRM	Query replication multiple
FPR	False-positive error rate
FNR	False-negative error rate
SQL	Structured query language
DBA	DB administrator
LAN	Local area network
PAG	Periodic accesses graph
ST	Session tracking
SE	Session evaluation
SE+	Session evaluation with lengths' partitioning
MCVs	Most common values
NDVs	Number of distinct values
JVM	Java virtual machine

ABSTRACT

Asmaa Sallam. Ph.D., Purdue University, May 2019. Anomaly Detection Techniques for the Protection of Database Systems against Insider Threats. Major Professor: Elisa Bertino.

The mitigation of insider threats against databases is a challenging problem since insiders often have legitimate privileges to access sensitive data. Conventional security mechanisms, such as authentication and access control, are thus insufficient for the protection of databases against insider threats; such mechanisms need to be complemented with real-time anomaly detection techniques. Since the malicious activities aiming at stealing data may consist of multiple steps executed across temporal intervals, database anomaly detection is required to track users' actions across time in order to detect correlated actions that collectively indicate the occurrence of anomalies. The existing real-time anomaly detection techniques for databases can detect anomalies in the patterns of referencing the database entities, i.e., tables and columns, but are unable to detect the increase in the sizes of data retrieved by queries; neither can they detect changes in the users' data access frequencies. According to recent security reports, such changes are indicators of potential data misuse and may be the result of malicious intents for stealing or corrupting the data. In this thesis, we present techniques for monitoring database accesses and detecting anomalies that are considered early signs of data misuse by insiders. Our techniques are able to track the data retrieved by queries and sequences of queries, the frequencies of execution of periodic queries and the frequencies of referencing the database tuples and tables. We provide detailed algorithms and data structures that support the implementation of our techniques and the results of the evaluation of their implementation.

1 INTRODUCTION

Cybercrimes committed by malicious insiders are among the most significant threats to systems [1]. Being an extremely important asset to organizations, databases (DBs) have been identified as the most vulnerable systems to insider threats according to recent insider threat reports [2]. The deficiency of strategies and solutions for the protection of data is the main reason why insider threats in the form of data breaches and leaks are rising.

DB systems have strong authentication mechanisms to ensure that the systems' users have proper credentials. Once a user is authenticated, access control mechanisms determine which DB objects can be read and modified by the user. However, authentication and access control mechanisms are unable to detect data misuse attempts by insiders who have proper privileges to access the data. They are also unable to detect a masquerader who has succeeded in stealing the credentials of a legitimate user of the system.

Anomaly detection (AD) is considered an effective approach for detecting data misuse scenarios by insiders and masqueraders. AD techniques tailored for DB systems detect deviations from the normal behavior that may indicate possible attacks. Such techniques rely on profiles that represent the normal users access patterns to the data. Profiles are based on historical data representing past interactions of the users with the monitored DB system.

In this thesis, we discuss AD techniques for securing the contents of relational DBs from insider threats. Although the problem of the mitigation of insider threats is challenging and its solution requires combining different techniques [3], our view is that an AD system that works at the DB layer, i.e., at the data source, is a promising approach towards detecting attacks by malicious insiders. This view is based on the following observations.

1. Studies indicate that deviations from normal behavior are indicators of possible insider attacks. Since AD is about building models to characterize the nor-

mal user behavior and then using such models as baselines for comparing user activity, AD is suitable for detecting indications of insider attacks.

2. DB access is performed through a standard query language (SQL) with well-understood and well-documented semantics. It is, therefore, feasible to baseline behavior at the DB layer, as opposed to doing so at the network or operating system layer, where the diversity of mechanisms and protocols for data transfer creates complexity that often confuses conventional intrusion detection systems.

3. Monitoring the potential disclosure of confidential data is most effective when done as closely as possible to the data source. Therefore, the DB layer is the most suitable place for detecting early signs of insider attacks.

4. The DB layer already has a thorough mechanism in-place for enforcing access control based on subject, i.e., users and applications, credentials. Additional information on the subject requesting the data, such as the role ID and IP address, is instrumental in detecting early signs of ex-filtration [4].

Our techniques are designed for the detection of five anomaly scenarios, which are not detected by the existing related work.

1. A query that retrieves a dataset whose size exceeds the normal size. Our approach for the detection of this anomaly scenario is to capture both syntactic and semantic features of the queries. We rely on the optimizer's output plan-trees to extract the semantic features of queries under inspection in order to avoid the execution of these queries for the purpose of AD.

2. Changes in the frequencies of execution of periodic queries. Such type of queries may be used for backup purposes and executed automatically to retrieve and save large-size datasets. A periodic query issued by a user who has the access privileges required for the query execution is considered normal according to role-based access control (RBAC); however, the execution of this query at a time different from the scheduled backup time is anomalous. To detect this anomaly scenario, our techniques automatically detect the periodic queries existing in past DB logs and track the frequencies and times of the execution of the periodic queries.

3. Queries that repeatedly retrieve specific data tuples. By comparing the result-sets of the execution of the same queries, the issuer is allowed to track updates to the contents of the retrieved tuples. To detect this anomaly scenario, we capture the normal rates of retrieval of the data tuples stored in the monitored DB. We consider exceeding any of these rates by one or more queries anomalous.

4. Sequences of queries that retrieve datasets whose sizes exceed the normal sizes. To detect this anomaly scenario, we propose capturing and tracking the sizes of data retrieved by queries during intervals of different lengths and during sessions of users' connection to the monitored database management system (DBMS).

5. Queries that read from tables at a rate higher than the normal rate. Such queries are an indication of attempts to retrieve large portions of the tables and/or to infer the distribution of the retrieved data. To detect this anomaly scenario, we record in the profiles the rates of execution of queries upon the DB tables. We consider exceeding any of these rates by one or more queries anomalous.

The design of our techniques takes into account several challenges associated with the problem of monitoring DB access.

1. The system must be able to monitor different commercial DBMSs and integrate with logging tools provided as part of commercially available security information and event management (SIEM) systems. To address this challenge, we present a system to detect, alert on, and respond to anomalies in DB access. The system's design is specifically tailored for commercial relational DBMSs.

2. The AD techniques must be able to monitor different types of data access, e.g., from users, application programs and internal DB maintenance, and variations in data access patterns. Our techniques consider different types of queries executed by users and application programs and can detect different kinds of changes in the access patterns including changes in the amount of retrieved data, and the frequencies of execution of queries and retrieval of data tuples.

3. The AD techniques must detect the anomalous queries before their result-sets are shown to the issuers. They must have good run-time performance in order

to minimize the impact of AD on query processing times. To address this challenge, we assure that the queries are inspected before their evaluation, whenever possible. In case the execution of a query is essential for its inspection such as for monitoring excessive retrieval rates of the data tuples, our methods rely on result-set pipelining to speed up the detection of anomalies.

The remainder of this document is organized as follows. Chapter 2 contains a review of the existing literature. In Chapter 3, we discuss the design and development of an architecture that directly supports AD in commercial DBMSs and present techniques for the detection of data ex-filtration attempts on relational databases. Chapters 4, 5 and 6 focus on the detection of temporal insider threats. In Chapter 4, we present techniques for tracking the execution of periodic queries in DBMSs and for the detection of related anomalies. In Chapters 5 and 6, we present techniques for the detection of temporal data ex-filtration attempts. We also discuss a system and a technique for the detection of anomalies in the rates of tuples retrievals in Chapter 6. Chapter 7 discusses potential future work.

2 STATE OF THE ART

In this chapter, we survey existing techniques and systems for the detection of anomalies in DB access. The different techniques have adopted diverse approaches for representing users actions. Each such approach has shown the capability to flag early signs of specific types of attacks. Our focus in this chapter is to elaborate on the key features of each technique, discuss the attack types it can help detect and suggest possible extensions to the proposed work.

This chapter is organized into three sections. In Section 2.1, we give an overview about the categories of information that can be inferred from users actions and the methodologies usually followed by existing systems and techniques for performing AD. We also discuss the different types of sources of queries and the implications imposed on systems and techniques which consider each type of query. In Section 2.2, we review the prominent AD systems and techniques.

2.1 Scope and Methodologies

2.1.1 Features Space

Features used for AD in database describe different aspects of queries that can be issued by users. There are four main categories of features:

1. *Syntax-based features.* Syntax-based features are extracted from the syntax of SQL queries and are used to describe queries' structures. Examples syntax-based features of a select query are the query's command type, range tables, i.e., tables that are referenced by the query, projection list, i.e., the attributes that appear in the query's result-set, and attributes referenced in the where-clause. Syntax-based features can quickly be extracted from queries as they only require parsing the queries' strings and traversing the resulting parse trees.

   AD that relies on syntactic features is useful in the detection of masquerading attacks. A masquerader is an insider or an outsider who succeeds in stealing the credentials of a legitimate user account [5]. The stolen credentials can be used for malicious purposes such as silently snooping on the DB. A masquerader usually has no knowledge about the access patterns of the true account owner and is thus unlikely to perform actions consistent with the account owner's typical behavior. This results in significant differences between the structure of queries executed by the masquerader and the normal queries. Such differences can be detected by syntax-based AD. Syntax-based AD is also able of detecting SQL injection attacks as these attacks too result in changes in the structures of the where-clauses of normal queries.

2. *Data-centric features.* Data-centric features are based on an analysis of the data in queries' result-sets. Example data-centric features of a query are statistics on the values of the projection list attributes, the volume of the query's result-set, and the raw tuples that correspond to the values in the result-set rows.

   The extraction of the data-centric features of a query can be costly if the execution of the query, parsing the query's result-set rows, or matching these rows to the raw tuples in the monitored DB is required. However, data-centric AD is able of detecting more sophisticated attacks compared to syntax-based AD. Data harvesting attacks that involve the extraction of data whose sizes exceed the normal or viewing data records that are out of the scope of the job functions of an insider are example attacks that can be detected by data-centric AD.

3. *Context-based features.* Context-based features describe the context in which queries are executed. The IP address, location, user ID and role of the issuer of a query are example contextual features of the query.

4. *Temporal features.* Temporal features are computed based on the time-stamps of execution of queries. Example temporal features are the order of execution of the individual queries in a sequence of related queries, the aggregate sizes of result-sets of a query sequence and the periodicity of a query or a group of queries.

   AD that relies on both temporal and data-centric features is useful in the detection of attempts to track updates on data tuples and of data aggregation threats [6]. Data aggregation is a data harvesting attack performed by executing multiple queries; each of which retrieves small portions of the target data-set.

2.1.2 Methodologies and Sources of Queries

The existing techniques and systems have considered two main sources of the queries executed against the monitored DB:

1. *Queries executed by DB tools.* DB systems provide their users with user interface (UI) tools that the users can utilize to interact with the systems. Example functions provided by a UI DB tool are user authentication, receiving user queries on the data and presenting queries result-sets in user appealing views. Example DB tools that can be used to interact with PostgreSQL[1] systems are pgAdmin[2] and psql[3].

   Queries executed by DB tools are ad hoc; as a result, the exact syntax of such queries cannot be predicted by the AD system. Data mining techniques are usually employed by AD systems that consider this type of queries in order to infer the users access patterns and match new queries to the learned models. Statistical methods are also employed to capture and track the temporal aspects of the queries executed through the use of DB tools.

2. *Queries executed by application programs.* Application programs are a different source of queries, which impose two types of constraints that govern queries syntax:

   (a) The structures of expected queries encoded in a program through the use of strings or prepared statements[4] is a static constraint.

   (b) The exact syntax of queries and their order is a dynamic constraint determined at run-time based on user inputs.

   AD systems that capture dynamic programs constraints require the use of sophisticated techniques to profile the programs executions paths and to follow

[1]https://www.postgresql.org/
[2]https://www.pgadmin.org/
[3]http://postgresguide.com/utilities/psql.html
[4]https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

the program's flow and compose the expected queries based on users inputs; however such systems provide finer profiles and better AD accuracy compared to systems that only capture static programs constraints.

2.2 Prominent Techniques and Systems

2.2.1 Syntax-based AD

Kamra et al. [7] propose a syntactic approach for the detection of anomalies in DB access. SQL queries are represented in the form of quiplets of attributes of one of three different granularities: coarse, medium and fine.

Two application scenarios that are based on the contextual features of queries have been considered. The first scenario is referred to as role-based AD. Role-based AD assumes that information on the roles of the issuers of queries is present in both the training logs and at the time of inspection of queries. In this scenario, the naive Bayesian classifier using the maximum aposteriori (MAP) rule is applied. To build the classifier, statistics on tables and attributes references are computed based on the quiplet representations of the training queries and stored in roles profiles. Given a new user query, role-based AD computes the probability that the query is issued by the users of all the roles of the monitored DB system; the query is considered anomalous if the probability that the query is executed by the role of the issuer is less than the probability that it is executed by any other role.

The second scenario is referred to as unsupervised AD. Unsupervised AD considers the case when role information is not present or incomplete and thus the AD system builds profiles of the individual users. During the training phase, unsupervised AD uses a standard clustering algorithm to form clusters of the quiplet representations of the training queries maintaining a mapping between each user and the clusters to which his/her queries belong. During the detection phase, a query z is inspected by first finding the cluster to which the query belongs (C_z), and then applying one of two methods to determine if the query is anomalous:

1. Use a classifier to determine if C_z is one of the clusters of the issuer.
2. Apply a statistical test to determine if z is an outlier to C_z.

The experimental evaluation is performed on both synthetic and real datasets [8]. Assuming that the data-sets do not contain anomalies, anomalies are simulated by *negating*, i.e., changing, the role ID, in case of role-based AD, and the user ID, in case of unsupervised AD, of the issuer of a query under inspection. Although the results of the evaluation show that the syntactic approach has high AD accuracy, the rate of generation of false-positive alarms in the case of real data-sets is high ($\sim$17-19%). This rate is considered unacceptable for DBs that receive large query streams. Another major problem in the experimentation is that the methods for simulating attacks are insufficient for proving that the syntactic approach will actually work for real attacks.

The proposed approach has several limitations.

- The MAP rule applied for role-based AD does not consider the case when queries are common among the different roles. In the case of a query under inspection that has been executed by the users who belong to different roles based on the training logs, the naive Bayesian classifier will be biased towards the role that has higher number of executions of the query.

- Role-based AD does not consider the case when one user belongs to multiple roles at the time of the execution of a query. This problem is common to the methods that rely on role membership for grouping users profiles [9–13].

2.2.2 Data-centric AD

Mathew et al. [14] propose modeling the users' access patterns by profiling the data points that the users access. Their approach relies on the observation that queries syntax alone is a poor representative of users intents, in contrast to the data accessed by the queries. They propose representing queries in the form of S-Vectors that encode statistics on the the data retrieved by the queries from the columns of the monitored DB. Statistics that represent a list of values extracted from a string type column are the values count and the number of distinct values in the list; whereas statistics that represent a list of values extracted from a numeric column are the maximum, minimum, mean, median and standard deviation of the values. The profile of a user or a role is a cluster of the S-Vectors of queries executed by the user or the users of the role.

During the detection phase, a query is considered anomalous if it is different from the profile of the issuer. Since it is not possible to consider all the data values in a large result-set of a query in order to compute its representative S-Vector, the authors propose two methods to approximate the S-Vector of a query:

1. Compute the S-Vector of the query based on the initial k rows in the query's result-set, or

2. Compute the S-Vector of the query based on random k rows in the query's result-set.

The second method is more suited for ordered columns.

The experimental evaluation is performed using different machine learning algorithms such as SVM, naive Bayesian classification and decision trees. The results of the evaluation show that the data-centric approach is superior to the syntax-based approach in detecting data harvesting attacks. However, the data-centric approach has a high rate of false alarms ($\sim$22%).

2.2.3 Temporal AD

Mazzawi et al. [15] propose algorithms for comparing users activities for both self-consistency, i.e., consistency with previous patterns of access by the same user, and global consistency, i.e., consistency with past actions of similar users. The purpose of training is to build two types of models:

1. *Self consistency models.* Self consistency models include:

   (a) *A rarity model.* For each user and every atomic action, e.g., SQL command type, seen in the training logs, the rarity model stores the probability of appearance of the action in a new time frame. For a user u_i and an atomic action e_j, such probability is denoted as $b_j^{(i)}$ and calculated by dividing the number of time frames during which u_i performed e_j by the number of time frames the user u_i was active.

   (b) *A volume model.* The underlying distribution of the number of occurrences of an atomic action by a user is assumed to follow the log normal distribution. The parameters of such distribution for each user-action pair is computed such that the likelihood of appearance of the set of actions by the user as observed in the training data is maximized; the computed parameters are stored in the volume model based on the training logs.

(c) *An out of context model.* The out of context model stores information on the correlations between the actions performed by each user. The profiler follows the steps below to build the out of context model for a user u_i.

  i. Model each atomic action performed by u_i in the form of a vector, referred to as the appearance vector. The appearance vector of u_i and an atomic action e_j is a list of Boolean values whose length is equal to the number of time frames during which u_i has been active based on the training data; the value that corresponds to time frame k is set to 1 if u_i has performed action e_j during the k-th time-frame.

  ii. Employ the similarity-based clustering algorithm, Iclust, on all appearance vectors related to u_i; this groups atomic actions into clusters of actions that tend to occur together in the same time frames. Information on the resulting clusters is stored in the out of context model.

(d) *A new object model.* The new object model stores the mean and variance of the number of objects that have not been accessed by each user in time frames of length equal to one hour.

2. *A global consistency model.* The behavior of each user is modeled in the form of a vector referred to as rarity score vector. The rarity score vector of a user u_i is denoted as $b^{(i)}$ and computed as

$$b^{(i)} = (b_1^{(i)}, b_2^{(i)}, ..., b_r^{(i)}),$$

where r is the number of atomic actions seen in the training data and $b_j^{(i)}$ is the appearance vector of u_i and action action e_j. k-means clustering algorithm is then run on the rarity vectors of all users to form clusters of users of similar behavior. The centroids of the resulting clusters and the user-cluster membership information is stored in the global consistency model.

Algorithm 2.1: Computing the anomaly score of user u_i based on actions previously performed by the user.

```
1. $E_i = \{\}$
2. For each action cluster $C$ in the out of context model:
    2.1. $S = \{\}$
    2.2. Pick one representative action $e_j$ from actions in $C$
    2.3. Compute an anomaly score $s_i$ based on the current time frame count
         of $e_j$ and the learnt rarity and volume models
    2.4. Add $s_i$ to $S$
3. Build a histogram $H$ that represents the anomaly scores of actions in $E_j$
   based on the user's logs
4. Compute the final anomaly score as the percentage of actions scores in $S$
   that fall below the values in $H$
```

The anomaly detection phase (also referred to as the analysis phase) starts after training is complete. The activities performed by each user are analyzed and two anomaly scores are computed for each user:

1. *Self-consistency score.* The self-consistency score of a user u_i during one time frame is computed as the maximum of two scores:

    (a) *Score based on the new objects accessed by the user.* A positive anomaly score is computed if the count of new objects accessed by the user during the current time frame exceeds the mean value associated with the user in the new object model. This score is computed based on the upper bound computed by applying Chebychev's inequality on the mean and variance of new objects that is associated with the user's information in the new object model.

    (b) *Score based on actions previously performed by the user based on the training data.* Algorithm 2.1 shows the steps for computing this score.

2. Global consistency score. This is computed based on the cosine similarity distance between the vector that represents the user's actions during the current

time frame and the centroid of the cluster of the user that is stored in the global consistency model.

The paper shows results of the evaluation of the proposed techniques on synthetic datasets only as per non-disclosure agreements, it is not allowed to show results on real customer data. However, the techniques have been integrated with InfoSphere Guardium[5], the SIEM tool developed by IBM, and evaluated in practice. The tool was able to alarm on an unusual volume of accesses by a customer; upon further investigating the alarm, the alarm was confirmed to be an actual attack.

Three attack scenarios were simulated using synthetic data and the performance of the proposed tool was evaluated. An attacker was defined as a person having 20% of his daily transactions being malicious. The results of the evaluation show that the self-consistency model performed better in the detection of unusual dropping of tables and attempts to query huge tables, which result in a decrease in the DB system performance. The global consistency model was more suitable for the detection of masquerading attacks in which an attacker steals one user account and silently snoops on the DB by accessing random tables that are not the usual tables accessed by the true account owner.

Possible improvements on the proposed models would be to detect anomalies due to abnormal activities that span multiple frames. Continuous monitoring also seem to be more accurate and more efficient than the periodic monitoring approach adopted in the proposed tool.

2.2.4 Profiling Application Programs

A. IIDD: Integrated Intrusion Detection in Databases

Fonseca et al. [16] propose a tool named IIDD that analyzes transactions executed on DBs by application programs. IIDD extracts query templates from the strings of queries executed against the monitored DB by replacing all non-generic values in the strings, e.g., constant numerics and strings, with place-holders. Profiles are either built manually by a DB administrator, concurrently to the normal utilization of the program to be profiled, or during program testing.

[5]https://www.ibm.com/security/data-security/guardium

The profile of a program is stored in the form of a directed graph; nodes in the graph represent queries, and paths in the graph represent the order of execution of queries in transactions.

The proposed tool was evaluated using the TPC-W benchmark in addition to a real DB. The results show that the proposed tool produces zero false-positive errors, can accurately detect incorrect ordering of commands and changes in queries syntax, and has low impact on the response times to queries. However, the number of query templates inferred by the profiler is large; this indicates that the profiling approach produces many redundancies.

The proposed approach has a major drawback that it cannot capture the impact of user input on the sequence of queries. It is also not mentioned how IIDD could capture the ordering of commands.

B. DetAnom: A System for Profiling and Monitoring Database Access Patterns by Application Programs for Anomaly Detection

Rafiul, Bossi et al. [17, 18] propose DetAnom, which overcomes the drawbacks of IIDD. The main purpose of DetAnom is to detect attempts for tampering the code of application programs in addition to SQL injection attacks by detecting changes in queries syntax and the order of executions of queries. The design goals of DetAnom is three-fold:

- Minimizing the number of changes made to the program being profiled,
- Minimizing the impact of program monitoring on the performance of the program, and
- Achieving high AD accuracy with low rate of false alarms.

DetAnom relies on the Concolic testing approach[6] for building program profiles. Concolic testing combines symbolic and concrete program execution to provide as much coverage of the program as possible during testing. DetAnom uses the same approach to profile the control flow of programs and to find the SQL queries executed by a program in addition to the constraints that have to be satisfied before each query can be executed.

[6]https://en.wikipedia.org/wiki/Concolic_testing

The profile of a program built by DetAnom profiler is a directed graph similar to the one described in [16]. However, the profile produced by the DetAnom profiler associates each node of the graph with the constraints that control the execution of the query that the node represents; this information is not captured by the IIDD tool.

DetAnom captures input parameters from the JVM (Java virtual machine) by instrumenting the Java libraries that read user inputs. The decision to instrument the libraries rather than instrumenting the program's code was based on the fact that software modifications are usually restricted by license agreements between the utilizing company and the software company responsible for code development.

DetAnom monitors the program execution and, based on the program's profile and user inputs, DetAnom flags queries that are not expected to be executed by the program as anomalous.

DetAnom has been evaluated using three application programs developed by the authors. The test programs have different numbers of unique queries and nested code blocks. The results of the evaluation indicate that DetAnom introduces low overhead on the response times to queries and low overhead on the network as a result of sending the queries to the AD server to inspect.

A few remarks must be made on the experimentation methodology and results.

- The overhead added due to parsing complex expressions was not studied in the experimental evaluation.
- Long time ($\sim$4 days) was required to profile a medium-size application program that contains approximately 500 lines of code. The resulting profile only covered 20% of the program's code. The authors attributed the slow down to the inefficient Concolic testing library they employed for profiling the program.

C. Profiling Web Applications

Valeur et al. [19] propose techniques for the detection of attacks on backend DBs accessed by web applications. They focus on the detection of three types of attacks:

1. SQL injection attacks, which allow an attacker to inject strings into SQL statements for the purpose of executing additional queries that expose senstive data or maliciously alter the DB,

2. Cross-site scripting attacks, which allow the execution of client-side code in privileged contexts, and

3. Data-centric attacks, which allow an attacker to insert data in the DB that are not in the expected values ranges.

Their approach is to parse each query, extract its tokens and infer the tokens data types. They employ a parser that references the DB schema to detect the names of tables and attributes in addition to the data types of attributes. This information is also used in finding constants whose source is user inputs and inferring their expected data types and formats. After each query is parsed, a feature selector component transforms the query into a query skeleton by replacing all tokens marked as constant with empty placeholders.

During the training phase, the skeleton of a query is used to update the current models. Whereas, during the detection phase, the skeleton of a query is used to look up the profiles for a similar query and compute an anomaly degree based on the difference between the query and the model; an alarm is generated if the difference exceeds a certain threshold.

Valuer et al. proposed several statistical models for describing constants of different data types. For example,

- String constants models describe the expected lengths, character distributions and prefixes and suffixes of string type constants, and

- Enumeration constants models are used to describe constants that can be one of a finite number of options. This type of constants is common in web applications forms in which a user of a form selects one value from a drop-down menu.

The evaluation of the proposed techniques indicates that they are capable of the detection of four simulated attacks:

- An SQL injection attack that aims at resetting the passwords of many users,

- An SQL injection attack that aims at enumerating all the DB users,

- A parallel password guessing attack in which the attacker attempts to speedup password guessing by trying one password against a whole users DB in parallel, and

- A cross-site scripting attack in which the attacker executes a script that inserts values stored in the user's document.domain into a DB table accessible by the attacker.

The evaluation also shows zero false alarms generated when an attack-free data-set is checked and low overhead per query ($\sim$0.20-1.00 ms). The proposed techniques can be extended by taking into account the percentage of server code coverage during training in the model evaluation.

3 DETECTION OF ANOMALOUS QUERIES THAT RETRIEVE DATASETS WHOSE SIZES EXCEED THE NORMAL SIZES

In this chapter, we present AD techniques for the detection of anomalous SQL queries submitted to the monitored DBMS. Queries are inspected individually in order to detect mismatches between the users' access patterns and their associated profiles that are based on past access logs. Changes in the referenced DB entities, i.e., tables and columns, and the sizes of data referenced in queries from the stored profiles are considered anomalies that may need further investigation.

Our AD techniques are based on extracting both syntactic and semantic features from the queries' parse and plan trees. Query planning (optimization) is one stage in the query execution pipeline in which the plan for the query execution is generated based on estimated costs of the operations performed in the query. Since cost estimation depends on data statistics stored at the data dictionary of the planner, the execution of a query under inspection is not required by our AD techniques.

We choose among two machine learning techniques: classification and clustering for detecting anomalies in different scenarios. Classification is used when all user queries have associated role information. In this scenario, we consider the log records labeled data and group records with similar labels/roles in one class whose normal access behavior is learned by a classifier.

On the other hand, in the scenario when the roles of the users are unknown, we use clustering techniques for learning the users' access patterns.

We employ the naive Bayesian classifier and the multi-labeling classifier for classification. The naive Bayesian classifier applies directly to our AD problem and can be easily updated when changes to the monitored DB data statistics or to the users' access patterns are encountered. However, the naive Bayesian classifier does not operate properly when multiple roles have common access patterns. This is the reason why we incorporated the multi-labeling classifier, which is designed mainly to handle this case. We use the COBWEB algorithm for clustering user queries. COBWEB shows very accurate knowledge acquisition capabilities in our problem settings.

In order to make our approach work for different DBMSs, we use a parser and an optimizer that are different from the ones used in the monitored DBMS. A critical issue in our approach is how to import in our AD system schema information and data statistics from the monitored DBMS dictionary. We describe our approach to address this issue for different commercial DBMSs, such as Microsoft SQL Server and Oracle. We performed extensive evaluation of the AD system we developed to implement our AD techniques. The evaluation indicates that our techniques are very effective in the detection of anomalies.

The remainder of this section is organized as follows. In Section 3.1, we present the architecture of the system. We discuss the internal representation of the queries features and the methods required for extracting such features in Section 3.2. In Section 3.3, we illustrate the statistics that are required to be available to the anomaly detector in order to support the feature extraction process and the methods for importing such statistics from the catalogs of different commercial DBMSs. In Sections 3.4 and 3.5, we describe the use of classification and clustering by our techniques to detect anomalies. We discuss the results of the experiments for the evaluation of the proposed techniques in Section 3.7. Section 3.8 concludes the chapter and discusses future work.

3.1 Architecture that Directly Supports AD in Relational Databases

The AD techniques we propose in this chapter can be employed by any organization that uses a commercial DBMS to manage its DB. We refer to the DBMS and the associated DB that are being monitored as the target DBMS (T-DBMS) and the target DB (T-DB), respectively. We refer to the component that contains the implementation of our techniques as the A-Detector and to the component that implements the methods for training the A-Detector as the Profiler.

The system operates in two phases: the training phase, which is done off-line, and the detection phase during which queries are inspected by the A-Detector to detect access anomalies.

During the training phase, the Profiler processes past logs of queries by the users of the T-DBMS. When information on the roles activated by the users at the time of executing the training queries is present in the log, the Profiler aggregates profiles of users who belong to the same role in order to form roles' profiles, which are more manageable than a large number of profiles for the individual users. The Profiler uses an SQL parser to parse the training queries and extract their syntactic features. In order to extract the semantic features of the training queries, the Profiler executes the training queries on a mirror of the T-DB; we refer to this DB as the mock DB (M-DB). Either a snapshot of the T-DB at the start time of the training phase or a complete log of the T-DB queries is thus required.

During the detection phase, all connection and query requests sent by the users or applications to the T-DBMS are intercepted by an SQL proxy and relayed to a mediator component. The mediator is responsible for coordinating between the different system components.

When given a new input user query, the mediator sends the query to an A-Detector component, which is responsible for the inspection of user queries and detecting and logging mismatches between the queries and the training profiles.

For the purpose of query inspection, the A-Detector employs a query parser and optimizer. Since these DB components require access to the schema of the T-DB and statistics on the data stored at the T-DBMS, we employ a component for importing the data required by the A-Detector's DB components into its internal catalog; we refer to this component as the schema and statistics importer (SSI). The SSI checks for updates in the data statistics stored at the T-DBMS after the execution of every few queries that result in changes in the data stored in the T-DB. Figure 3.1 illustrates the architecture of DBSAFE; that is the system in which we implemented the techniques described in this chapter.

3.2 Data Representation

Training queries and the queries being analyzed during the detection phase are represented similarly in the form of quadruplets of fields. For the sake of simplicity, we represent a generic quadruplet using a relation of the form: (c, P_R, P_A, S_R). A description of the fields of a quadruplet q_p that represents a query q is as follows.

- c stands for command and represents q's command type and can be either Select, Insert, Update or Delete.
- P_R stands for projected relations and contains information on the relations R whose attributes are projected in q, i.e., whose data appears in q's result-set.

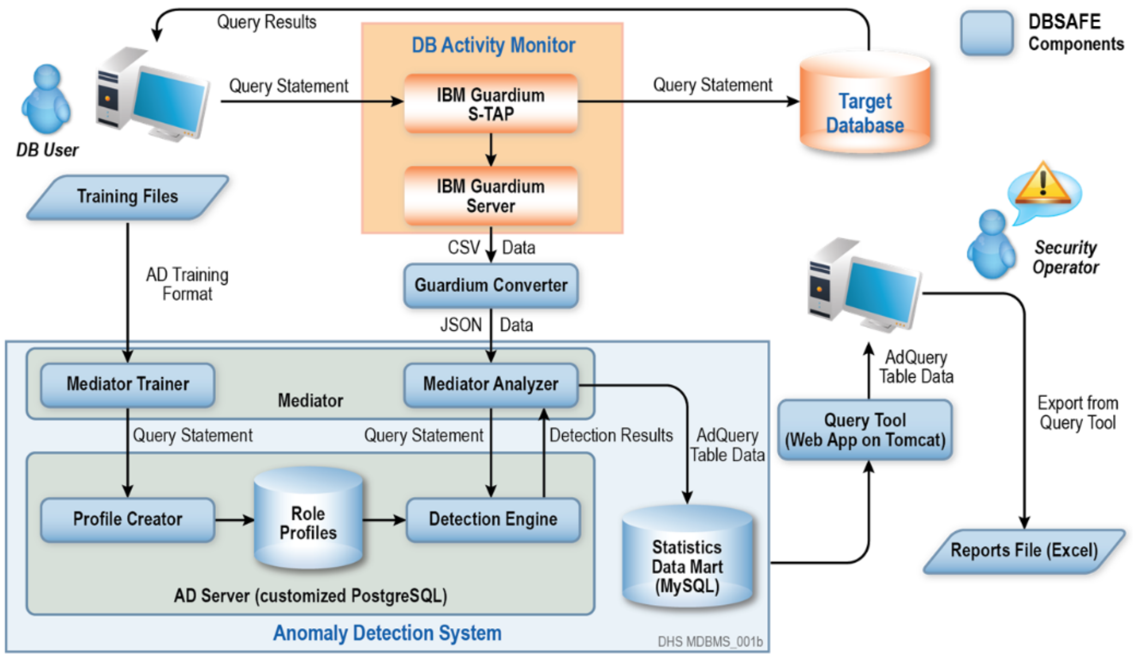


Figure 3.1.: DBSAFE Architecture

P_R is a binary array whose length is equal to the number of relations in the T-DB. Entries in P_R that correspond to R have TRUE value; other entries have FALSE values.

- P_A stands for projected attributes and contains information on the attributes projected in q. P_A is a two-dimensional array whose rows correspond to the T-DB relations and columns correspond to the T-DB attributes. Entries that correspond to the attributes projected in q are TRUE and other entries are FALSE.

- S_R stands for selectivity of relations and contains the selectivity levels of the relations referenced in q. We consider four selectivity levels: l_0, which represents relations that are not accessed in the query, and l_s, l_m, and l_l, which represent the ranges [0, 0.33[, [0.33, 0.66[, and [0.66, 1], respectively.

It can be noted that all the components of the quadruplet can be obtained from the query's parse-tree except for S_R, which can be obtained by executing the query or by parsing the query's plan-tree. In Section 3.3, we present an approach for extracting the selectivity of relations from the plan-tree of a query.

Compared to the profile format described in [7], the quadruplet representation of queries does not include information on the attributes referenced in the WHERE-clauses of the queries. We decided to omit this data from the representation as the result of experimentation presented in the same paper show that they do not have a positive effect on the accuracy of AD.

To illustrate the quadruplet representation, consider an example DB representing a university that consists of several colleges. Each college has a number of departments to which students are affiliated. The DB contains three relations: Students, Departments and Colleges. The relation Students contains a foreign-key attribute named s_deptID that references Departments.d_ID. The relation Departments has a foreign-key attribute d_c_ID that references Colleges.c_ID. The data stored in the relations is shown in Tables 3.1a, 3.1b, and 3.1c. Table 3.1d contains example queries that reference the DB tables and the corresponding quadruplet representation of these queries.

3.3 Extraction of Selectivity Information

3.3.1 Algorithms

An important issue in the design of our system is how to extract information that characterizes the data retrieved by queries during the detection phase. We propose an approach for extracting this information based on the plan trees of queries prepared by the planner. We use this approach to avoid the requirement of executing queries under inspection. The goal is to reduce the time required for performing AD and provide low impact on the response times to input queries.

Since a DBMS usually has many execution plans to compose the result-set to a query, the planner's job is to find a good plan for evaluating the query by ordering the query operators and selecting the method for executing each operator. The output of the optimizer is a tree-structured plan for query execution; each node in the tree is an operator and the input(s) to this operator is the result of the child nodes. The choice of the planner on which plan to select is based on the cost of the query plan which is usually measured as the number of secondary storage page reads and writes.

Table 3.1.: Quadruplet representation of queries in role-based and user-based AD

(a) Students

s_ID	s_lastName	s_d_ID
100	Alex	1
101	Daniel	2
102	Luke	3
103	Mike	1
104	Neil	4
105	Zachary	3

(b) Departments

d_ID	d_c_ID	d_name
1	1	CS
2	1	IT
3	2	ECE
4	2	EE

(c) Colleges

c_ID	c_name
1	Science
2	Engineering

(d) Quadruplet representation of example queries

Query	**Quadruplet**

Q1. Select all the students whose last names start with the letter 'A'.
 	
SELECT *
FROM Students
WHERE s_lastName LIKE 'A%'
	c = 'SELECT'
	$P_R = [1, 0, 0]$
	$P_A = [\ [1, 1, 1],$
	$[0, 0, 0],$
	$[0, 0, 0]\ ]$
	$S_R = [l_s, l_0, l_0]$

Q2. Print all the IDs, last names and department names of the students.
 	
SELECT s_ID, s_lastName, d_name
FROM Students, Departments
WHERE s_d_ID = d_ID
	c = 'SELECT'
	$P_R = [1, 1, 0]$
	$P_A = [\ [1, 1, 0],$
	$[0, 1, 0],$
	$[0, 0, 0]\ ]$
	$S_R = [l_l, l_l, l_0]$

Q3. Print the IDs and last names of the students affiliated to the college of science.
 	
SELECT s_ID, s_lastName
FROM Students, Departments, Colleges
WHERE s_deptID = d_ID and d_c_ID = c_ID and c_name = 'Science'
	c = 'SELECT'
	$P_R = [1, 0, 0]$
	$P_A = [\ [1, 1, 0],$
	$[0, 0, 0],$
	$[0, 0, 0]\ ]$
	$S_R = [l_m, l_m, l_m]$

In the course of estimating the cost of a query plan, the planner also estimates the cost of each operator using statistics on tables and columns stored in the DB catalogs. For example, consider the query Q3 shown in Table 3.1d. Different plans for executing this query are possible. Figure 3.2 shows two such plans. However, the planner may favor Plan B over Plan A as the former includes smaller sizes of intermediate data.

In order to extract the selectivities of the relations referenced in an input query, we process the plan-tree of the query in a top-down recursive manner. The goal is to determine per-table selectivities using the selectivity estimates computed by the planner for the operators, i.e., internal nodes, of the plan-tree. As we process the plan tree we compute the cardinality of each node. The selectivity of a table that is referenced in the query is set to the minimum cardinality of the nodes in the path from the root of the tree to the leaf node that references this table, as computed by our algorithm, divided by the estimated table cardinality. The estimated cardinality of a table is obtained from the statistics maintained in the A-Detector's catalog.

We refer to the algorithm we propose for extracting per-table selectivities as Selectivity-Estimate. The input to the algorithm is a plan tree of a query that is the output of the planner of PostgreSQL. While parsing a plan tree, the algorithm processes the nodes differentiating between two types of nodes.

1. Single-input nodes. A node that has one input represents a unary operator. Four unary operators exist in PostgreSQL: Sequential-Scan, Index-Scan, Materialize, and Hash operators.

    (a) The Sequential-Scan and Index-Scan nodes are processed by Algorithm Selectivity-Estimate similarly due to the similarity of the nodes structures. However, the two nodes have different purposes. An operator of type Sequential-Scan or Index-Scan has a raw table input and may have an associated condition that filters the rows of its input. The Sequential-Scan operator indicates that the execution of the plan retrieves the rows of the input table from the source of the data in the rows. On the other hand, the Index-Scan operator indicates that, during the plan execution, portions of the rows of the input relation are retrieved from a secondary source of data represented by an index on the table's attributes.

When Algorithm Selectivity-Estimate encounters a Sequential-Scan or an Index-Scan node, it sets the cardinality of this node to the output cardinality of the node that is extracted from the node's attributes.

(b) Algorithm Selectivity-Estimate processes the Materialize and Hash operators similarly. The Materialize operator creates in-memory storage for an intermediate result. The purpose is to avoid performing multiple reads of the same data or computing a result multiple times during the course of the execution of a query. A Hash operator is the left child node of a Hash-Join operator and its function is to insert the rows in its input into a hash-table to perform the Hash-Join algorithm.

Algorithm Selectivity-Estimate computes the selectivity of a node that represents a Materialize operator or a Hash operator by obtaining the result of dividing the expected number of rows in the node's output by the number of rows in the node's input; this selectivity is further used to compute the number of rows accessed in the tables that the node is part of the path of their processing.

2. Two-input nodes. A node that has two inputs represents a join operator. We consider two cases:

   (a) The input to the join operator is two raw tables (R_1 and R_2), there is a primary-key-foreign-key relationship between the two tables and the join is based on the attributes of this relationship.

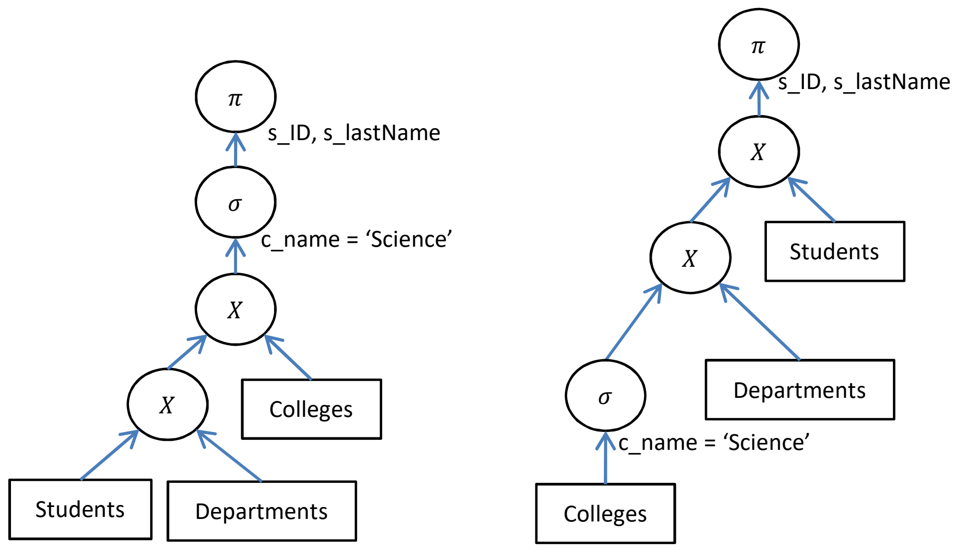
   If R_2 has a foreign-key that references R_1, all the rows of R_2 are assumed to appear in the result of the join operator, excluding the rows that have null values in the foreign-key attributes of R_2.

   If the number of distinct values in the foreign-key attributes of R_2, referred to as ndv_2, is available in the catalog, the selectivity of the equi-join node with respect to the sub-tree that references R_1 is set to ndv_2 divided by the cardinality of R_2.

   If the value of ndv_2 is not present in the catalog, the selectivity of the equi-join node with respect to the sub-tree that references R_1 is set to 1.

If the join has a filter condition, the cardinality of the node seen by the left and right subtrees is reduced by the ratio of the cardinality of the output divided by the number of rows of R_2.

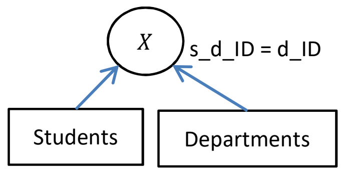
(b) In all other cases. The selectivity of both nodes is reduced by the ratio $\min(\frac{n_o}{n_1}, \frac{n_o}{n_2})$, where n_o is the estimated number of rows in the output of the current node, and n_1 and n_2 are the number of input rows to the current node. If the number of output rows of the node is larger than the value computed for one of its input nodes, the number of rows related to such node is set to the number of output rows of the node.



(a) Plan A (b) Plan B

Figure 3.2.: Two possible optimizer plans for an example query

```
SELECT *
FROM Students, Departments
WHERE s_d_ID = d_ID
```



(a) Query (b) Query plan

Figure 3.3.: Optimizer's plan for a query containing an equi-join operator

3.3.2 T-DBMS Adapters

Our implementation of the A-Detector is based upon PostgreSQL DBMS. We use the parser and optimizer of PostgreSQL to generate the quadruplets of queries to be inspected. In order to employ these DB components, either the data in the T-DB or the schema and statistics of such data should be available at the A-Detector. We chose to mirror and maintain the schema and statistics on the T-DB data in the A-Detector's catalog in order to avoid the replication of the T-DB data.

As mentioned earlier, supporting the different types of commercial relational DBs is one of the goals of the design of our system. It is thus required to consider the different administrative commands for the extraction of the schema and statistics stored at the T-DB. An adapter for a DBMS may compute some statistics on the data from scratch when such statistics are not supported by the DBMS; otherwise, if the statistics are stored in the T-DBMS's catalog, the adapter directly imports such statistics in order to save the resources required for re-computation.

In terms of implementation, we developed T-DBMS adapters for Oracle and SQL Server DBMSs. In both adapters, we used the views provided for the DB administrator (DBA) to obtain the schema information before starting the detection phase.

In order to maintain the mirrored schema, updates to the T-DB schema upon the interception of a DDL statement at the proxy are applied to the A-Detector's catalog. The adapters maintain height-balanced histograms, most common values (MCVs) and number of distinct values (NDVs) at the A-Detector's catalog to be used by PostgreSQL optimizer. In the following, we present details on the methods required to maintain these statistics by Oracle and SQL Server adapters.

A. Oracle T-DBMS Adapter

Oracle DBMS creates different types of histograms. Oracle determines the type of histogram associated with the data of an attribute depending on the DBMS version and the parameter values configured by the DBA. We next discuss the methods required for converting the different histogram types to the height-balanced histogram type used by PostgreSQL optimizer.

1. Height-Balanced Histograms. Oracle DBMS versions earlier than 12c create height-balanced histograms for the data stored in a DB column if the NDVs in

this data is greater than the number of buckets expected to be in the histogram; the number of histogram buckets is a configuration parameter provided by the DBA at the time of statistics gathering. The buckets of a height-balanced histogram have equal size, i.e., they represent equal number of rows.

Although, this histogram type has the same structure as the histograms maintained by PostgreSQL, Oracle adapter has to perform some transformations before saving the histogram at the A-Detector's catalog. The reason is that an Oracle height-balanced histogram not only encodes the distribution of values in an attribute's data, but also encodes the MCVs of this data. When a value is represented by more than one bucket, Oracle keeps record of this value in one bucket only; it can be inferred from the bucket's attribute named "ENDPOINT_NUMBER" that this value is repeated a number of times more than the bucket size by finding if one or more succeeding buckets are not present in the histogram data. In this case, it can be assumed that the value of "ENDPOINT_VALUE" of the bucket spans the buckets whose information is missing from the histogram. Oracle adapter thus copies the height-balanced histograms from Oracle to PostgreSQL catalog by directly copying the buckets that do not span more than one bucket, adding buckets for the values that span more than one bucket and considering these values MCVs.

2. Hybrid Histograms. The information on the number of repetitions of the MCVs that is inferred from the missing buckets of a height-balanced histograms is inaccurate in some cases. As a result, Oracle employs hybrid histograms instead in Oracle 12c. Like height-balanced histograms, hybrid histograms have equal-size buckets. However, hybrid histograms distribute values such that no value occupies more than one bucket and store an attribute called "ENDPOINT_REPEAT_COUNT" for each endpoint (bucket) in the histogram; this value indicates the number of times the endpoint value is repeated. Oracle adapter retrieves the MCVs and their frequencies directly from the buckets endpoints and their repeat count values, respectively.

3. Frequency Histograms. There is no direct conversion from frequency histograms to height-balanced histograms. In this case, the MCVs and histograms are not retrieved from Oracle's data dictionary, but computed by querying the data in

the relations in the T-DB. The creation of a frequency histogram in Oracle can be avoided if the number of buckets of the histogram provided as input by the DBA at the time of statistics gathering is chosen to be greater than the NDVs in the data of the attribute to be analyzed.

B. SQL Server T-DBMS Adapter

SQL Server DBMS stores frequency histograms for attributes' data. Therefore, all statistics on tables and attributes that are used by PostgreSQL optimizer can be imported directly from SQL Server statistics except histogram data. SQL Server adapter executes the necessary aggregate queries on the T-DB and uses the result-sets to build the height-balanced histograms expected by the A-Detector's optimizer.

3.4 Role-based AD

In this section, we describe our methodology for AD when the T-DBMS has RBAC in-place. Role information is thus associated with the queries in the training logs and queries to be inspected during the detection phase.

The purpose of training in this application scenario is to aggregate logs that belong to users of the same role and infer and record the behavior of the roles' users in profiles. Mismatches between the behavior of the users during AD and the profiles of the roles they belong to are considered anomalies that may be the result of data misuse intents.

We address the role-based AD problem as a classification problem for which we used three different types of classifiers: the binary classifier, the naive Bayesian classifier, and the multi-labeling classifier. Details on how the different types of classifiers are used in AD are provided in the rest of this section.

3.4.1 The Binary Classifier

Given an input query and the role of its issuer, the binary classifier's task is to decide if such query has been issued previously by any user of this role based on the training logs. The training profiles contain data that speeds the training logs look-up. The profile of a role contains a Boolean value for each attribute, which indicates whether the attribute has been projected in one or more training queries issued by the users of the role. An input query is considered anomalous if the value associated with any of the query's projection list attributes in the training profile of the role of the issuer is FALSE.

The binary classifier has pros and cons.

- The time required for constructing the training profiles and for query inspection is very low as queries are required to be parsed only for feature extraction.
- Training profiles can reside in-memory, since the space required for storing such profiles is very low. Using in-memory profiles leads to speeding up AD for input queries; therefore, AD has low impact on the response time to queries.
- The training profiles of the binary classifier can be manually revised by administrators. An administrator can simply toggle the Binary values associated with attributes. This advantage is crucial for the correct operation of the classifier when insufficient data is present for constructing the training profiles.
- On the other hand, the binary classifier cannot capture how the attributes are related in the training queries, i.e., which attributes are referenced together in queries.
- Also the binary classifier cannot rule out a training query that is not repeated in the logs. Anomalous queries present in the training log thus significantly affect the accuracy of profiles.

3.4.2 The Naive Bayesian Classifier (NBC)

The NBC has proven to be effective in many practical applications such as text classification and medical diagnosis; this is mainly because of the low computational cost for training and detection that results from assuming that attributes considered in classification are independent. Bayesian rules of probability can then be applied with a decision rule to perform the classification. The Maximum-A-Posteriori decision rule (MAP) is most commonly used with the NBC; the MAP decision rule results in correct classification as long as the true class of an instance is more probable than all others. In the rest of this section, we describe the general principles of the NBC and show how it can be applied to our use case.

In supervised learning, a classifier is a function f that maps input feature vectors $x \in X$ to output class labels y_i, where $i \in 1...C$, X is the feature space, and the length of the vector x is n. Our goal is to learn f from a labeled training set of N input-output pairs.

One way to solve this problem is to learn the class-conditional density $p(y|x)$ for each value of y and to learn the class priors $p(y)$. Bayes rule can then be applied to compute the posterior $p(y|x) = \frac{p(x,y)}{p(x)}$. Since the goal is to assign a class to the new instance x, a decision rule is then applied. By applying the MAP decision rule, x is assigned to the most probable class, i.e., x is assigned to the class y that maximizes $p(y|x)$.

Assuming the feature vector $x = [a_1, a_2, ..., a_n]$, the Bayesian and MAP rules can be applied as follows based on the previous results in [7]:

$$y_{map} \; \alpha \; \text{arg-}\max_{y_j \in Y} p(y_j) \prod_i p(y_j|a_i) \tag{3.1}$$

The NBC directly applies to our anomaly detection framework by considering the set of roles in the system as classes and the log file quadruplets as observations. The number of attributes of the system is $|P_R.P_A^T| + |P_R.S_R^T| + 1$. For example, if the database has 2 tables and each table has 3 attributes, each table may or may not be present in the query and each attribute may or may not be referenced; therefore, the number of possible combinations of the presence of the tables and attributes is 2 * 3 and each table can have one level of selectivity in the query; the type of the query constitutes an additional attribute of the query. The previous equation can be rewritten as:

$$r_{map} = \text{arg-}\max_{r_j \in R} p\left(Q|r_j\right) \tag{3.2}$$

$$p\left(Q|r_j\right) = p\left(r_j\right) p\left(c|r_j\right) \prod_{i=1}^{N} p\left(\mathcal{P}_{\mathcal{R}}[i].\mathcal{P}_{\mathcal{A}}^{\mathcal{T}}[i]|r_j\right) p\left(\mathcal{P}_{\mathcal{R}}[i].\mathcal{S}_{\mathcal{R}}^{\mathcal{T}}[i]|r_j\right) \tag{3.3}$$

The probability that each role sends the input query is computed using Equation 3.3. In the equation, $p(r_j)$ is the prior probability, that is the probability that the role r_j sends a query, and is equal to the number of queries executed by the users of r_j according to the training log divided by the total number of queries in the log. The rest of Equation 3.3 constitutes the posterior probability that the role r_j executes the input query. $p(c|r_j)$ is the probability that a user holding the role r_j executes a query that has the same command type as the input query. $p\left(\mathcal{P}_{\mathcal{R}}[i].\mathcal{P}^{\mathcal{T}}_{\mathcal{A}}[i]|r_j\right)$ is the probability that a user who belongs to role r_j executes a query that projects columns of table i that appear in the query. $p\left(\mathcal{P}_{\mathcal{R}}[i].\mathcal{S}^{\mathcal{T}}_{\mathcal{R}}[i]|r_j\right)$ is the probability that a similar user executes a query that accesses table i and has the same selectivity as the input query. The previous two probabilities are computed for each table in the T-DB and combined in the computation of the posterior probability using multiplication $(\prod_{i=1}^{N})$ due to the independence assumption. The output of the classifier (r_{map}) is computed using Equation 3.2 and is equal to the role that has the maximum probability of executing the input query; R is the set of roles in the T-DBMS. After r_{map} is computed by the NBC, its value is compared to the actual role of the user. If they are identical, the query is considered normal; otherwise the query is considered anomalous.

The per-table selectivity component of a query (S_R) is computed based on the optimizer's output plan of the query that is based on the statistics stored on the tables in the T-DBMS catalog; therefore, when the statistics of a table change, the per-table selectivity of the training queries that reference this table may also change and should be updated accordingly.

To be able to incrementally execute the required updates, the mediator keeps a hash-table that maps the identifier of each table to the list of training queries corresponding to this table. Each query has the ID of the role of the user who submitted the query and the selectivity of the table in this query.

When the mediator receives new statistics on the data of one table, it refers to the hash-table to find the queries that reference the table; the mediator sends a negative-update message to the A-Detector, which has the form $(-, r_i, t_j, s_k)$; such a message informs the A-Detector that the profile of the role whose ID is r_i should be updated by decrementing the count of queries in which the selectivity of t_j is s_k; this value is denoted as $C(t_j, s_k)$. The mediator also sends similar messages for each table in the query and adds the query to a redo-list. The redo-list contains queries that will be further processed by the mediator as follows. The mediator requests that the A-Detector computes per-table selectivities of the tables in each of the queries in the list and sends positive-update messages to the A-Detector as follows. If a query q_j executed by a user of role r_i references a table whose identifier is t_k and the selectivity of this table in this query is s_{jk}, the mediator sends the A-Detector a message of the form $(+, r_i, t_k, s_{jk})$. The A-Detector responds to this message by incrementing $C(t_k, s_{jk})$. It can be noted that by using the positive and negative updates, the A-Detector does not need to update other features stored in the profiles, but only the selectivity of tables in queries in the redo-list.

3.4.3 The Multi-Labeling Classifier (MLC)

An important problem to consider when choosing the type of classifier to use for AD is the case when there is overlap between the access patterns of roles. Classifiers like the NBC that employ the MAP rule do not produce accurate results in this case. One could use a well-known concept in data mining called feature selection to exclude query features that have no or very low significance on the results of classification. This concept is not suitable for our problem setting because excluding features when performing the detection is not appropriate from the security aspect as these features will be unmonitored.

Disregarding the fact that the feature selection concept is not suitable for AD, we experimentally evaluated the accuracy of AD when using the maximum relevance minimum redundancy (mRMR) feature selection technique [20] to solve the problem of common queries between roles. In our experiments, we used such technique to compute the entropy of each feature and the implementation by Peng et al. [20]. An important challenge in using this technique is choosing which attributes to consider based on the computed entropy values. We developed two selection policies for this purpose. The first is to choose the relevant attributes based on a threshold value τ; all attributes that have entropy below τ will thus not be considered in AD. The other policy is to consider only the N features that have the maximum entropy values. Both policies have the problem that the choice of their parameters depends on the particular training set. Therefore, the results for our experiments on feature selection were not promising.

A family of classifiers, called the multi-labeling classifiers, is better-suited to address the problem of roles with overlapping access patterns. The idea behind these classifiers is to associate each training query instance with more than one role if it is a common query to the roles. Accordingly, an input query can be labeled with more than one role during detection.

The existing methods for multi-labeling classification are categorized into two classes: algorithm adaptation and problem transformation. In algorithm adaptation methods, some learning algorithms are extended in order to handle multi-labeled data, whereas in problem transformation, the multi-label learning task is mapped onto one or more single labeling problems. Binary Relevance (BR) [21], Binary Relevance+ (BR+) [22] and Label Power Set (LP) [23] are example problem transformation methods.

We decided to use the BR approach for multi-labeling classification, since BR has the advantage of having low computational complexity compared to other multi-labeling methods. Given R T-DBMS roles, BR employs R binary classifiers. The complexity of BR is thus R * O(C), where O(C) is the complexity of each binary classifier. On the other hand, BR has the disadvantage that it fails to consider label dependency. This disadvantage, however, is not considered a limitation for our approach, since the label independence restriction holds among the T-DBMS roles.

Using the BR multi-labeling approach, the Profiler constructs R binary SVM [24] classifiers during training that represent queries by R T-DBMS roles. Each binary classifier is trained using the queries of a specific role and all other training queries issued by the other roles; a classifier can thus distinguish between whether the role it represents could possibly send an input query. To classify a new query, BR employs the classifier that represents the role of the query issuer..

3.5 User-based AD

In the application scenario when no role information is associated with user queries, we consider the problem of AD as an unsupervised learning problem for which we used clustering techniques to summarize user queries.

To build the training profiles, we first employ a standard clustering algorithm to form clusters of the training records of all the T-DBMS users. For each T-DBMS user, we record the clusters to which his/her queries belong. We refer to such mapping as the user-clusters mapping.

When given an input query to inspect during the detection phase, the A-Detector employs the same clustering algorithm used in training to find the cluster to which the query belongs. Using the user-clusters mapping, the A-Detector also finds the clusters associated with the query issuer. If the cluster of the input query is one of the issuer's clusters, the query is considered normal and anomalous otherwise.

To decide on which clustering algorithm to use, we evaluated the efficacy of k-means [25], Expectation-Maximization (EM) [26], Farthest-First [27, 28] and COBWEB [29] clustering algorithms. Eventually, we decided to use COBWEB, since it showed high knowledge acquisition capabilities and accurate AD results. COBWEB is a conceptual clustering algorithm, which builds a classification tree based on the training observations.

COBWEB builds the training tree incrementally starting with an empty tree and adding each training record one at a time. The structure of the tree in terms of branching depends on the category utility (CU) evaluation metric, which is used to measure the quality of the resulting tree. The addition of a training record is the process of classifying the record by descending the tree along an appropriate path, and performing one of the following operations at each level: classifying the record with respect to an existing class, creating a new class, combining two classes into a single class, or splitting a class into several classes. The decision on which operation to perform is taken on a level-basis by considering the operation that results in the best CU score.

3.6 Taxonomy of Anomalies

In this section, we review the types of anomalies presented in [14] and discuss which types can be flagged by our AD techniques. A query is characterized by its result-set, which contains data whose structure is referred to as result-set schema. Two queries are considered different if they have different result-set schemas or if the data in their result-sets are statistically different. A query is considered anomalous by AD if it is different from the training queries. The differences between queries can be classified into three types as follows.

1. Type-1: Different schema. A query that has a schema that is different from the schemas of the training queries is considered anomalous. For example, if the training queries of a role reference some attributes of one relation, while a

detection query references other attributes of the same relation or the attributes of other relations, the detection query should be considered anomalous.

Our AD technique can detect this type of anomaly as the result-set schema of each query is recorded in its quadruplet representation.

2. Type-2: Similar schema/different data. Anomalies of this type are generated due to queries that have data that is statistically different from the result-sets of the training queries. This category has two subtypes:

   (a) Type-2a. This type includes queries that have similar syntax. An example of this type is represented by the queries:

       SELECT * FROM Students WHERE d_deptID = 1

       and

       SELECT * FROM Students WHERE d_deptID != 1.

   (b) Type-2b. This type represents the case when the normal and anomalous queries have different syntax. As an example, consider the two queries:

       SELECT * FROM Students WHERE s_ID = 1

       and

       SELECT * FROM Students WHERE 1.

       Our AD approach can flag type-2 queries only when the sizes of the result-sets of the normal and anomalous queries are different.

3. Type-3: Similar schema/similar data. This category is divided into two subtypes:

   (a) Type-3a. This type includes queries that have different syntax and similar semantics, i.e., user intent. An example of this type is represented by the queries:

       SELECT * FROM Students WHERE s_ID = 1

       and

       SELECT * FROM Students WHERE s_ID = 1 and s_deptID IN (SELECT d_ID FROM Departments).

       Our AD techniques do not consider this difference anomalous, which is considered the correct AD result.

(b) Type-3b. This type includes the case when the queries have different syntax and different semantics. As an example, consider the two queries:

Select * FROM Students WHERE s_ID < 100

and

SELECT * FROM Students.

Detection queries of this type can only be considered anomalous by our AD techniques if they have different result-sets sizes from the training queries.

3.7 Experimental Evaluation

We evaluated the performance of the techniques proposed in this section by performing three sets of experiments.

1. In the first set (Set A), we evaluated the accuracy of AD for the Binary and the naïve Bayesian classifiers in the detection of changes in the access patterns to the DB entities. The experiments are performed in collaboration with Northrop Grumman (NG) Corporation using the different resources that a company can offer.

2. In the second set (Set B), we evaluated the performance of the NBC and the MLC for different misuse scenarios that represent some of the anomaly types discussed in Section 3.6.

3. Similar to the second set of experiments, the third set (Set C) evaluates the performance of the NBC and the MLC, but instead of using the synthetic datasets prepared by us, we used the OLTP-Benchmark workloads.

In the rest of this section, we discuss the results of the three sets of experiments and compare the advantages and disadvantages of using each type of classifier in AD.

3.7.1 Set A – Experiments in Collaboration with NG

The T-DB used for evaluation has the schema of a government medical DB used to train doctors and other personnel. The data records of this DB are artificially-generated. Three roles and seven users exist in the T-DBMS. The total number of queries used to construct the profile of a role ranges between 10,000 and 12,000 queries.

The evaluation is performed in two independent runs. The length of an evaluation run ranges between 2 and 6 hours. In each run, seven concurrent users access the T-DB with mostly normal queries and relatively much less misuse queries.

The test engineers involved in the evaluation are divided into three teams: the blue team, the red team and the white team. The blue team is responsible for monitoring the system and protecting sensitive data using standard security techniques leveraging the prototype software. The red team performs the normal actions associated with the production system. Select members of the red team attempted to misuse sensitive data periodically during the evaluation test run and cover up the misuse activities. The identities of these members are unknown to the blue team. The white team monitors the evaluation process and gathers metrics for performance analysis. Figure 3.4 illustrates the test environment.

The red team is supplemented by two background workloads injected via the Apache JMeter load-testing tool. One workload consists entirely of normal queries; those are queries consistent with the tasks/activities of roles and are part of the training log. The second workload consists entirely of queries previously developed by the evaluation team to ex-filtrate data. The ex-filtration queries fall into two categories. The first category consists of select queries that retrieve very sensitive attributes of personally identifiable information, such as social security numbers and passport numbers. The second category consists of insert queries that create new actors/users in the system.

Before employing the AD techniques, the blue team inspected the queries received by the T-DB in order to evaluate the necessity of using automated AD. The result indicated that the blue team could not recognize any ex-filtration attempt, despite using a small set of SQL queries and lengthy time between successive queries. We then measured the accuracy of AD by the classifiers during two evaluation runs.

Tables 3.3 and 3.4 report the results of the first and second runs, respectively. The columns named “Bayesian classifier with warnings” shows the accuracy of the NBC when queries that reference tables and attributes that have never been referenced by the users of the role of their issuer’s produced a warning and were considered anomalous. It can be noted that although the same set of SQL statements were

Table 3.2.: Description of training data used in the evaluation of DBSAFE

Training Database
- 30 Tables
- Avg. 18 columns per table
- Avg. 600,000 rows per table
- 3 roles (doctors, nurses, hospital administrators)
- 7 unique DB user IDs

Training Queries
- 10,000 - 12,000 queries per role

Evaluation Runs
- Approx. 2 - 6 hours per run
- 7 users per run (4 human, 3 automated via JMeter)
- Approx. 520 total queries per run
- Approx. 40 exfiltration attempts per run

sent to both classifiers, the two classifiers reported different numbers of statements processed.

The results of evaluation show that the NBC consistently outperformed the Binary classifier. Both classifiers perform well with respect to true positives at the cost of unacceptably high false positives. The cause of generation of most of the false positives is that the training logs did not have enough data about the attributes that are usually retrieved by queries that are considered normal. The result is that the A-Detector flags queries that are considered normal by the blue team members as anomalous. It is thus important to have training data that covers all the access patterns that are considered normal.

Another observation on the results is that the Binary classifier generates a higher number of false positives compared to the NBC. The reason is that the former is not able to deal well with the case in which the detection queries have some minor variations with respect to the training queries. For example, suppose that the training data contains the query "Select s_ID FROM students". If a user who belongs to this role executes the query "Select s_ID, s_deptID from students", the Binary classifier will classify this query as anomalous; on the other hand, the NBC will classify it as a normal query as long as this role has a higher probability of sending such query compared to the other roles.

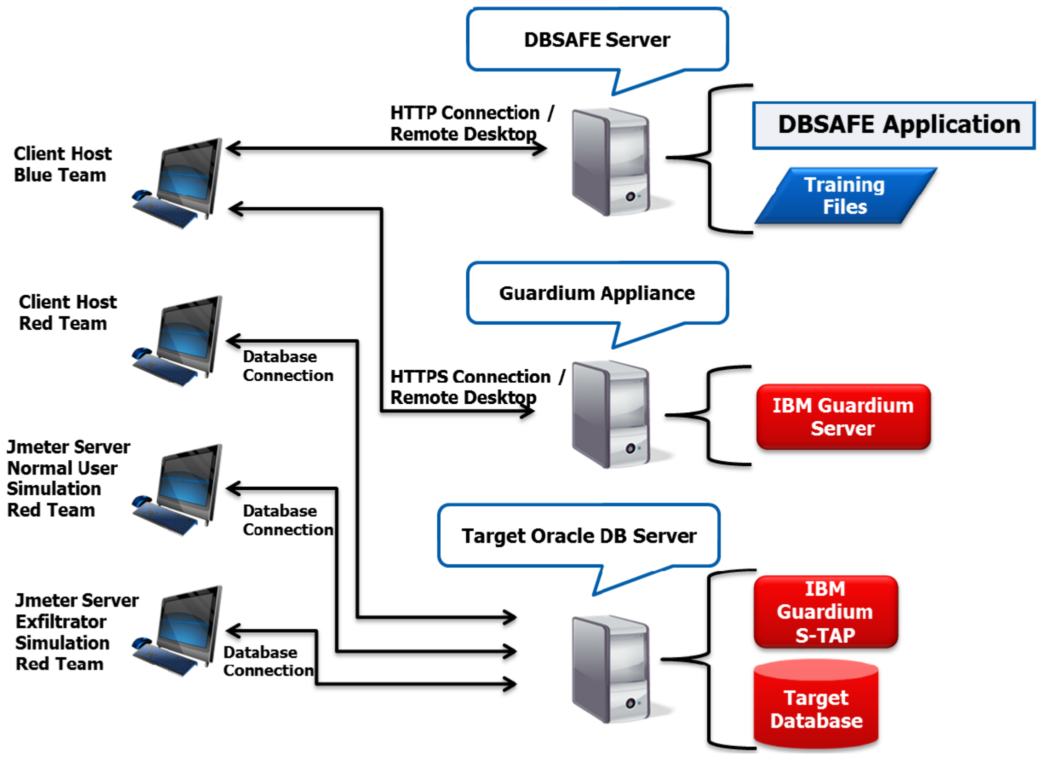


Figure 3.4.: DBSAFE evaluation environment

Table 3.3.: First run of evaluation of DBSAFE

	Actuals	NBC		NBC with Warnings		Actuals	Binary Classifier	
		Number	%	Number	%		Number	%
True Positives	38	26	68.42%	36	94.74%	42	37	88.10%
True Negatives	432	316	73.15%	305	70.60%	478	130	27.20%
False Positives	0	116	26.85%	127	29.40%	0	348	72.80%
False Negatives	0	12	31.58%	2	5.26%	0	5	11.90%
Total	470	470	200.00%	470	200.00%	520	520	200.00%

Table 3.4.: Second run of evaluation of DBSAFE

	Actuals	NBC		NBC with Warnings		Actuals	Binary Classifier	
		Number	%	Number	%		Number	%
True Positives	44	29	65.91%	43	97.73%	39	31	79.49%
True Negatives	414	339	81.88%	283	68.36%	359	106	29.53%
False Positives	0	75	18.12%	131	31.64%	0	253	70.47%
False Negatives	0	15	34.09%	1	2.27%	0	8	20.51%
Total	458	458	200.00%	458	200.00%	398	398	200.00%

3.7.2 Set B – Experiments using Synthetic Datasets

We show the results of evaluating the performance of the NBC, the MLC, and COBWEB clustering for AD using training and detection queries generated by us. The datasets consist of different patterns that describe the queries by the T-DBMS roles in case of role-based AD and users in case of user-based AD.

In each experiment, we use three quarters of the generated queries, chosen at random, as the training queries and the rest as detection queries. For measuring the false negatives, we generate the anomalous queries with the same distribution as the normal ones, but with negated role/user information; for example, in case of role-based AD, if the role of the query issuer is r, we generate the same query and assume that the issuer of the query belongs to roles existing in the system other than r. In time measuring experiments, we used a virtual machine that has 3 cores and 6 GB RAM and runs Ubuntu Linux .

In most experiments, we use the rates of false positives and false negatives as metrics for the accuracy of AD. However, we use the accuracy metric in case of the NBC, since it summarizes information on the false positives and false negatives results for the classifier by considering the percentage of false positives to be equivalent to the accuracy and the percentage of false negatives equivalent to the accuracy divided by the number of roles in the T-DBMS.

In addition to measuring the accuracy of AD, we compare the times required to perform AD on a query by the different classifiers. This metric is important as AD is usually executed during query evaluation and may result in higher response times to queries and lower throughput of the T-DBMS. In what follows, we describe the patterns represented by the evaluation scenarios and then show the results for role-based and user-based AD experiments.

A. Test Scenarios

In the first three test scenarios, we use a generated DB that contains 20 tables and 10 attributes per table. We measure the accuracy of detection for different numbers of training records (200, 500, 700, and 1000 records). In case of role-based AD, 10 roles exist in the T-DBMS and multiple users belong to each role. In case of user-based AD, 10 users exist in the T-DBMS. The rest of the discussion attributes queries to roles. The same discussion applies for the same scenarios used in experiments for evaluating user-based AD.

In the first access scenario (sc-1), the access patterns of roles do not overlap, i.e., each role accesses different tables and no two roles access the same table. Whereas in the second access scenario (sc-2), the users of the different roles may access the same tables, but no two users from different roles access the same attributes of one table. In the third scenario (sc-3), the different roles share access to the same attributes; when this case occurs in two queries executed by different users, the selectivity levels of the tables referenced in the queries in their result-sets are different.

In the fourth scenario (sc-4) and the fifth scenario (sc-5), we model the query access patterns of roles using the Zipf and reverse Zipf (R-Zipf) probability distributions. The Zipf probability distribution function (pdf) for a random variable X is defined as:

$$Zipf(X, N, s) = \frac{1/x^s}{\sum_{i=1}^{N} 1/i^s}$$

Where N represents the number of values X can take and s is the parameter characterizing the amount of skew present in the distribution. As s increases, the probability mass function (pmf) of the distribution accumulates towards the left elements. The R-Zipf distribution is the mirror of the corresponding Zipf plot with respect to a vertical axis at the median of the values of the random variable.

The T-DB used in sc-4 and sc-5 contains 20 tables; each table has 5 attributes. The T-DBMS has 4 roles. The first role normally accesses the T-DB tables using Select queries that have pdf Zipf(20, sr). It accesses the attributes with probability Zipf(5, sc). Similarly, the second role accesses the T-DB tables using Select queries, but the pdfs governing the distribution of these queries to tables and attributes are R-Zipf(20, sr) and R-Zipf(5, sc), respectively. The third and fourth roles issue Update commands only. The pattern of access of the third role to the tables and to the attributes follow Zipf(20, sr) and Zipf(20, sc), respectively. The pattern of access of the fourth role to the tables and to the attributes follow R-Zipf(20, sr) and R-Zipf(20, sc), respectively.

For scenarios sc-4 and sc-5, we measure the accuracy of AD with respect to changes in the amount of overlap between the selectivities of the tables in the detection queries; we also vary the parameters sr and sc of the Zipfian distributions of tables and attributes. In sc-4, we set *sc* to a low value equal to 0.5 and vary *sr* from 0.5 to 4.5 and the overlap between the selectivities of relations in queries from 100% to 0%. In sc-5, we set the value of *sr* to 0.5 and vary the value of *sc* between 0.5 to 4.5 and the value of the selectivity overlap from 100% to 0%.

The sixth scenario (sc-6) represents the case in which there is overlap in the training patterns of accesses to tables. We consider the range [5%, 35%, +5%]. The seventh scenario (sc-7) is inspired by the second dataset used by Kamra et al. [7]. In sc-7, we assume a T-DBMS that contains 9 roles and a T-DB that contains 40 tables; each of which has 20 attributes. The access patterns of roles are shown in Figure 3.5. All roles execute Select queries only.

B. Results for Role-based AD

The chart in Figure 3.6a shows the AD accuracy for the NBC for scenarios sc-1, sc-2, and sc-3. In sc-1 and sc-3, the classifier produces accurate results for all numbers of training records; however, in sc-2, the detection is accurate only when 500 or more tuples are used for training. In this scenario, the classifier requires more training queries to infer the patterns of access to the attributes.

Figures 3.6c and 3.6d show the accuracy of AD for scenarios sc-4 and sc-5. The results show high accuracy for values of *sc* and *sr* higher than 1.5 and values of selectivity overlap less than 80%.

The MLC shows 100% accurate detection in sc-1 and for training log lengths equal to or greater than 1000 records in sc-2. Unlike the NBC, which is less than 50% accurate in sc-6, the MLC yields 100% accurate results in this scenario as it can relate queries to multiple roles. This result is also supported by the experiments on sc-7 for which the evaluation results in case of the NBC and the MLC are shown in Figures 3.6e and 3.6f, respectively.

To determine the effectiveness of balancing the training data on the performance of the NBC, we consider the scenario when one T-DBMS role issues most of the queries in the training log and the other roles issue a small percentage (p) of the training queries. We vary the value of p between 5% and 20%, and measure the accuracy of AD by the NBC. Figure 3.6b shows that the NBC has poor performance for low values of p. The figure also shows that the accuracy is significantly enhanced when the training data is balanced before starting the detection.

Figures 3.6g and 3.6h show the time required by the NBC to perform AD for a query when different numbers of roles and different numbers of attributes are present at the T-DBMS. These factors affect AD as for each query under inspection, the NBC checks the probability that the query is issued by each T-DBMS role. This operation involves iterating over all the T-DB attributes and extracting the associated counts stored in the training profiles. The inspection time of a query is measured as the length of the interval between the receipt of the query by the proxy and the time when the AD result is ready.

Comparing the time required by the NBC and the MLC, the inspection time by the NBC is a few tens of milliseconds, while the inspection time by the MLC may reach one second. The MLC requires significantly more time than the NBC because the MLC builds multiple binary SVM classifiers whose number is equal to the number of roles in the T-DBMS. Moreover, the complexity of a single SVM classifier is much higher than that of the NBC.

In general, the MLC is more accurate and faster than the NBC. However, the NBC is more suitable for DBs whose data changes dynamically over time and for DBs that have highly dynamic schemas where adding and deleting roles and tables are frequent operations. Changes to the training data for the NBC can also be executed incrementally as described earlier. On the other hand, applying these changes to the model of the MLC may require rebuilding the model of the classifier, which is an expensive operation as shown in Figure 3.6i.

C. Results for User-based AD

We now show the accuracy of user-based AD for the different scenarios. The results of experiments show 100% accuracy for scenarios sc-1, sc-2, and sc-6, less than 10% false positives and 0% false negatives for all lengths of the training log in sc-3 and 0% false positives in sc-7. Figures 3.7a, 3.7b, 3.7c and 3.7d show the rates of false positives and false negatives for scenarios sc-4 and sc-5, respectively. Figure 3.7e shows the rate of false negatives for sc-7, which indicates a decrease in the error rate from 13% to 5%, when s increases from 0.5 to 4.5. The time required for the execution of AD is less than 10 milliseconds when the number of attributes projected in the query under inspection is less than 20.

It can be concluded from the experimental results that the proposed clustering algorithm can distinguish between the different roles when there are differences in the access patterns of these roles as shown in the results for sc-1, sc-2 and sc-3. It can also capture the overlap between the roles when this overlap is in references to tables. However, the overlap in the access to attributes cannot be captured by the clustering algorithm. If such training pattern is encountered in practice (this can be detected using cross-validation), supervised learning should be used instead by creating a role for each user and using the MLC for classification. We thus plan to extend our approach to detect such cases and recommend the proper approach that can provide accurate AD.

3.7.3 Set C – Experiments using the OLTP-Benchmark

We report the results of experiments that we performed using the OLTP-Benchmark workloads. We decided to use the benchmark, since it is typically hard to gain access to real datasets. The OLTP-Benchmark is the reference benchmark for evaluating the performance of DBMSs, contains diverse datasets and workloads and thus serves the purpose of evaluation.

Since the benchmark does not have role definitions, we selected some benchmark workloads, defined T-DBMS roles, and attributed the queries generated by the scenarios of these workloads to one or more roles based on our understanding of the scenarios and the jobs of the defined roles. The workloads for which we could define roles and thus use in our experiments are: AuctionMark, Epinions, Seats, and TPCC. We defined two roles for each workload: workers and clients. The roles of the AuctionMark and Epinions workloads access common scenarios.

To setup our experiments, we ran scripts for the creation of the T-DB schema, role definition, and the insertion of the T-DB records. The metrics that we used for evaluation are the number of true negatives (TN), the number of false positives (FP), the number of false negatives (FN), the number of true positives (TP), the true positive rate (TPR), the false positive rate (FPR), the true negative rate (TNR), the false negative rate (FNR), precision, accuracy, and F1-score. Based on these metrics, we computed the area under the ROC curve (AUC).

To measure the rates of false positives, we used 80% of the queries in the logs of previous runs of the benchmark for training and used the rest of the queries for detection. An anomaly that is generated for any of the detection queries is considered a false positive. To measure the rates of false negatives, we checked the result of the inspection of the log queries when the role/user information of these queries is negated. Since all these queries are considered anomalous, if the detection result indicates that a query is normal, this decision is considered a false negative.

Tables 3.6, 3.7 and 3.8 show the results of the experiments described above for the NBC, the MLC and the clustering algorithm, respectively. It can be noted from the results that the MLC has the best performance. The NBC has very high false positive rates for the AuctionMark and Epinions datasets. This result is expected because the roles have common access patterns for these workloads. On the other hand, user-based AD performs reasonably well on the AuctionMark workload. However, it has high false negative rate for the Epinions workload. The problem is that user-based AD cannot distinguish between the following two queries:

- SELECT avg(rating) FROM trust WHERE ...
- SELECT avg(rating) FROM trust, review WHERE ...

The two queries differ in one attribute that is the selectivity of the table review. The values of the corresponding attribute in the quadruplet representations of the first and queries are l_0 and l_s, respectively.

Table 3.5 shows the time required for training each of the NBC, the MLC and the COBWEB clusterer for each dataset in the benchmark. In general, as the size of the workload increases, the training time increases too. For the largest workload that contains about 26.5k training records, the training time does not exceed 3.5 minutes. Since training is done offline, the training rates that we observed are acceptable. The average time required to perform AD is 12 milliseconds for the NBC, 0.89 second for the MLC, and 17 milliseconds for the clusterer. In all cases, the detection time is less than a second and is thus acceptable to be part of the response time of a query.

3.8 Conclusions

In this chapter, we proposed a system for monitoring access to commercial DBMSs and presented techniques for the detection of anomalies in the access patterns to the DB objects and the sizes of data extracted from the monitored DB. Based on the

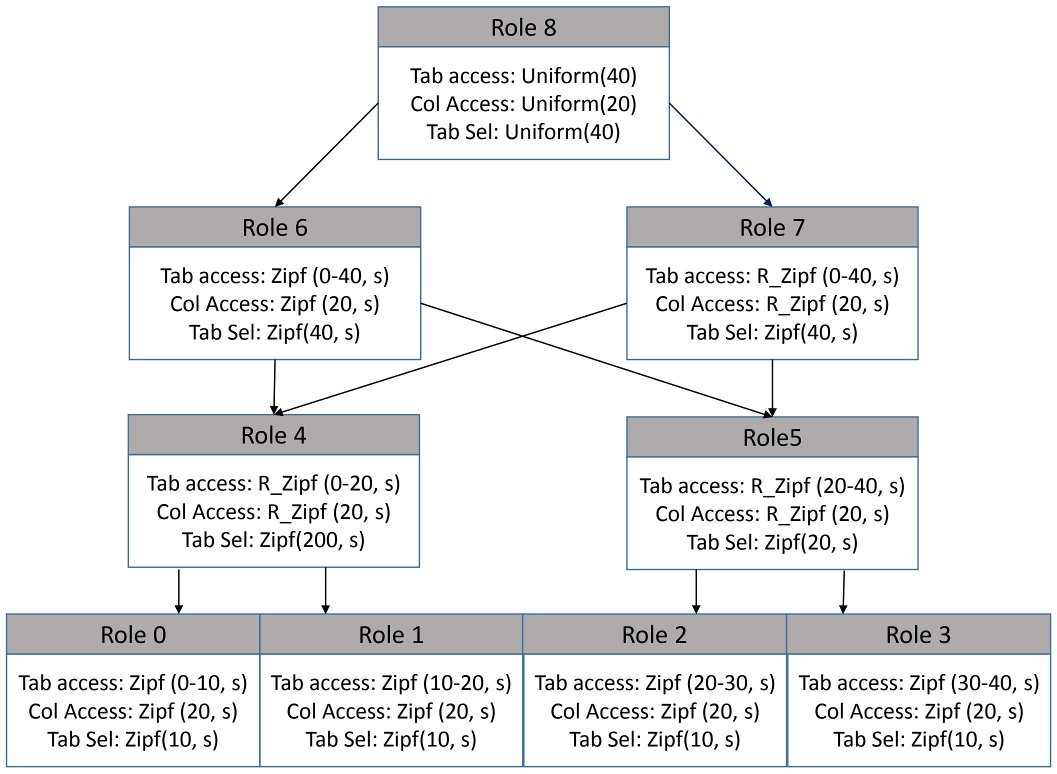
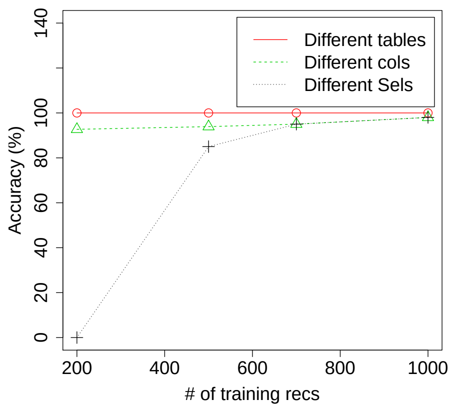


Figure 3.5.: Test case used for the evaluation of role-based and user-based AD

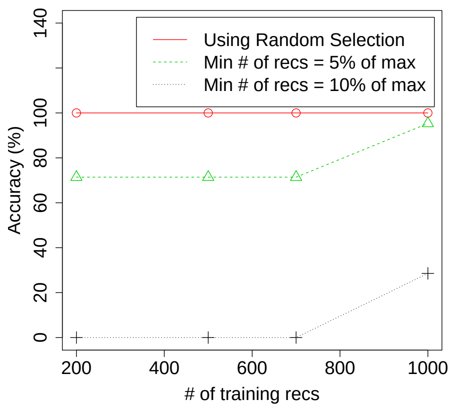
Table 3.5.: Training time (in mins) of role-based and user-based AD

Dataset	**NBC**	**MLC**	**COBWEB**	**Size (recs)**
Auctionmark	1.04	1.52	1.19	11.3k
Epinions	2.48	3.41	3.5	26.6k
Seats	1.83	3.35	4.32	18.1k
TPCC	0.85	1.13	1.01	7.4k

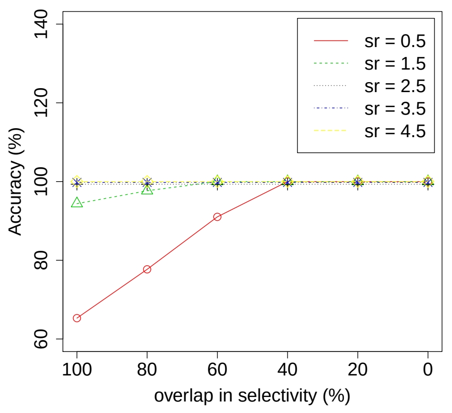
results of our experimental evaluation, our techniques can effectively detect deviations in data selectivities from the normal levels captured based on the training logs.



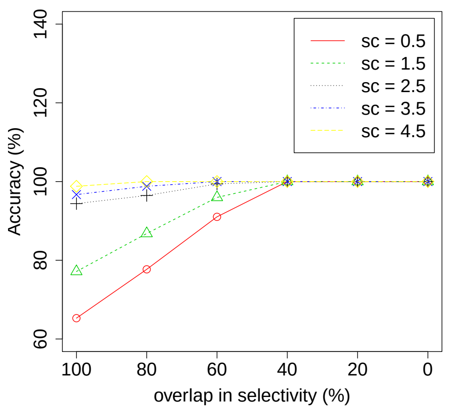
(a) Test cases 1, 2, 3 - NBC



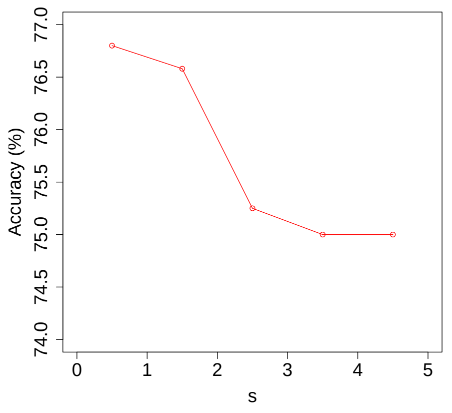
(b) Balancing training data - NBC



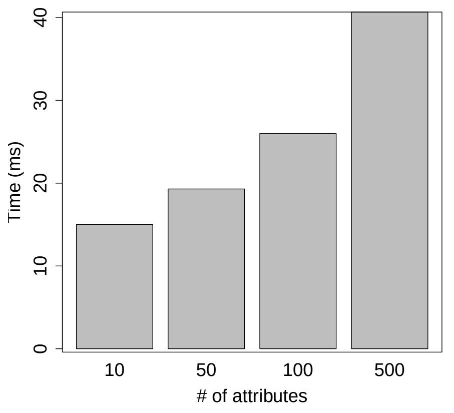
(c) Test case 4 - NBC



(d) Test case 5 - NBC

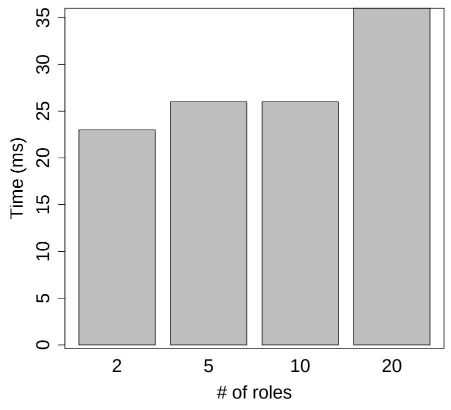


(e) Test case 7 - NBC

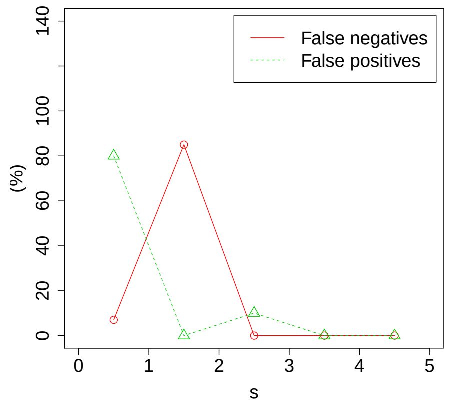


(f) Time per query - NBC

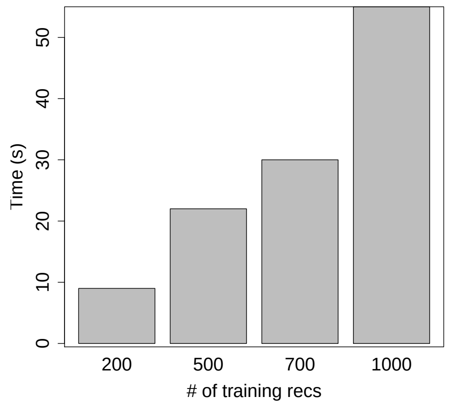
Figure 3.6.: Results of the evaluation of role-based AD



(g) Time per query - NBC

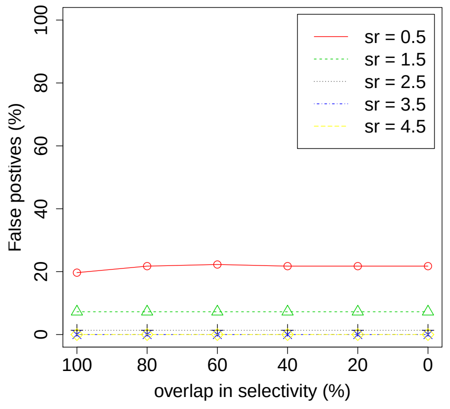


(h) Test case 7 - MLC

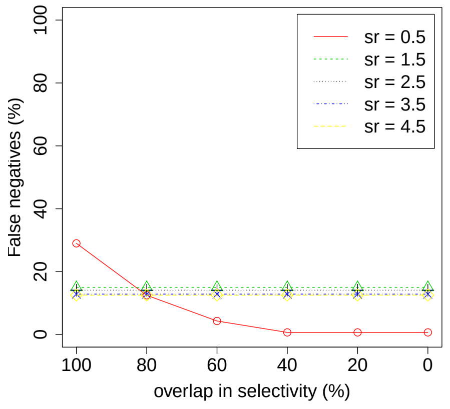


(i) Time to build training model - MLC

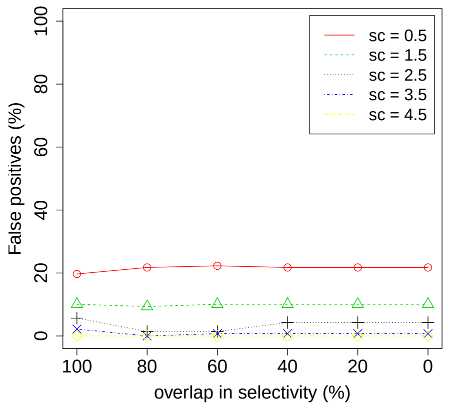
Figure 3.6(cont.): Results of the evaluation of role-based AD



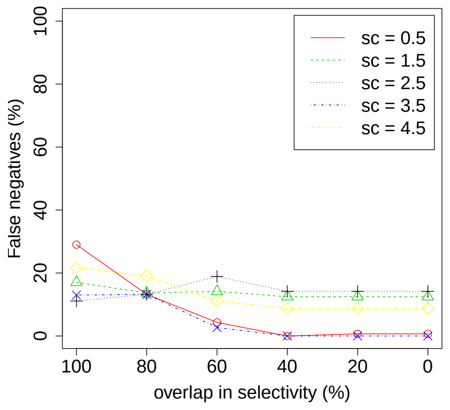
(a) Test case 4 - False positives



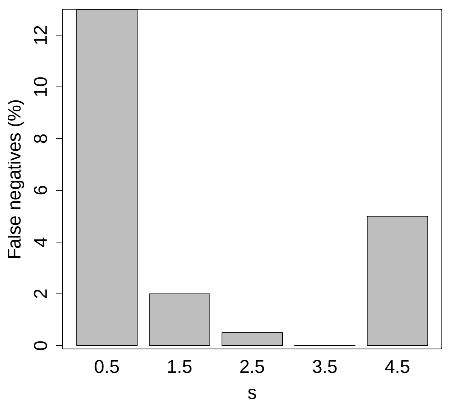
(b) Test case 4 - False negatives



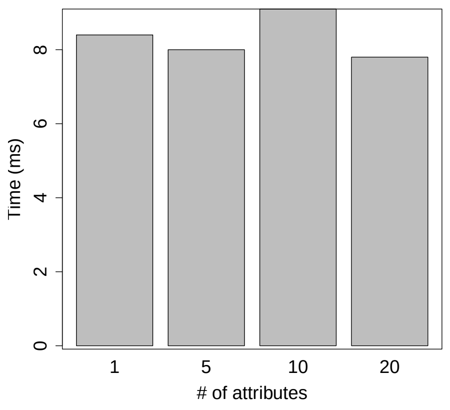
(c) Test case 5 - False positives



(d) Test case 5 - False negatives



(e) Test case 7 - False negatives



(f) Time per query

Figure 3.7.: Results of the evaluation of user-based AD

Table 3.6.: Results of the evaluation of the NBC used in role-based AD

	Confusion Matrix				**TPR**	**FPR**	**TNR**	**FNR**	**Prec**	**Acc**	**F1**
	TN	**FP**	**FN**	**TP**							
Auction Mark	470	125	151	1817	92.33	21.01	78.99	7.67	93.56	89.23	92.94
Epinions	9531	4736	570	3537	86.12	33.2	66.8	13.88	42.75	71.12	57.14
Seats	2581	95	259	7047	96.45	3.55	96.45	3.55	98.67	96.45	97.55
TPCC	3636	81	16	738	97.88	2.18	97.82	2.12	90.11	97.83	93.83
				Avg	93.195	14.985	85.015	6.805	81.2725	88.6575	85.365
				Std	5.27	14.86	14.86	5.27	25.92	12.28	18.92
				Conf	5.16	14.56	14.56	5.16	25.4	12.03	18.54

AUC = 88.96

Table 3.7.: Results of the evaluation of the MLC used in role-based AD

	Confusion Matrix				**TPR**	**FPR**	**TNR**	**FNR**	**Prec**	**Acc**	**F1**
	TN	**FP**	**FN**	**TP**							
Auction Mark	595	0	134	1834	93.19	0	100	6.81	100	94.77	96.47
Epinions	14267	0	0	4107	100	0	100	0	100	100	100
Seats	2676	0	0	7306	100	0	100	0	100	100	100
TPCC	3674	43	2	752	99.73	1.16	98.84	0.27	94.59	98.99	97.09
				Avg	98.23	0.29	99.71	1.77	98.6475	98.44	98.39
				Std	3.36	0.58	0.58	3.36	2.71	2.49	1.88
				Conf	3.29	0.57	0.57	3.29	2.66	2.44	1.84

AUC = 99.87

Table 3.8.: Results of the evaluation of the COBWEB used in user-based AD

	Confusion Matrix				**TPR**	**FPR**	**TNR**	**FNR**	**Prec**	**Acc**	**F1**
	TN	**FP**	**FN**	**TP**							
Auction Mark	595	0	226	1742	88.52	0	100	11.48	100	91.18	93.91
Epinions	14267	0	988	3119	75.94	0	100	24.06	100	94.62	86.32
Seats	2676	0	107	7199	98.54	0	100	1.46	100	98.93	99.26
TPCC	3717	0	7	747	99.07	0	100	0.93	100	99.84	99.53
				Avg	90.5175	0	100	9.4825	100	96.1425	94.755
				Std	10.86	0	0	10.86	0	4.02	6.19
				Conf	10.64	0	0	10.64	0	3.94	6.07

AUC = 99.53

4 MONITORING THE FREQUENCIES OF EXECUTION OF PERIODIC QUERIES

In this chapter, we present techniques for the detection of unexpected changes in the frequencies of execution of periodic queries. The methodology we follow for inferring the normal access frequencies is to process past logs of queries by the DB users in three stages. The first stage uses a standard periodicity detection algorithm to find the frequency according to which each query in the logs is consistently executed. The second stage determines the interval of time when each periodic query is expected to be executed. The third stage identifies the relationships between periodic queries that are executed together. This information is recorded in profiles upon which subsequent user queries are checked. A periodic query is considered anomalous if it is received at the T-DB at an unexpected time. The incorrect ordering of related periodic queries is also considered anomalous.

The techniques presented in this chapter complement the techniques discussed in the previous chapter. Using the techniques presented in this paper solely is insufficient as they are unable to properly inspect queries that are aperiodic. These queries can only be considered normal if they *resemble* the aperiodic queries in the training logs of the roles of the query issuers.

The design followed in this paper has various advantages. The architectural design of the system in which our techniques operate follows the system design presented in the previous chapter, which aims at supporting the operation of AD on any commercial relational DBMSs and their integration with existing security information and event management (SIEM) tools. Furthermore, the proposed techniques only require parsing and planning the queries under inspection and, as a result, can detect anomalies before the anomalous queries have side-effects on the DB being monitored. The proposed techniques can thus be employed as part of real-time alerting systems. To the best of our knowledge, our techniques represent the first attempt to mitigate temporal insider threats.

The rest of this chapter is organized as follows. In Section 4.1, we describe the anomaly scenarios that our techniques are able to detect and the architecture in which these techniques can be deployed. In Section 4.2, we discuss the features of queries that are considered by our techniques and the format we propose for the representation of such features. We then provide details on the training phase and detection phase algorithms in Sections 4.3 and 4.4, respectively. We describe an approach for the experimental evaluation of our techniques and the results of the evaluation in Section 4.5. Section 4.6 concludes the chapter and highlights potential future work.

4.1 Architecture and Scope of the work

In this section, we show a use case in which the techniques described in this chapter are effective in the detection of indicators of insider attacks. We also show the environment in which the implementation of our techniques operates.

4.1.1 Use Case

Consider a department store that uses a DB to manage customers and sales information. This DB contains three tables: *Customers*, *Sales* and *OldCustomers*. The table *Customers* contains sensitive information about the customers of the store, e.g., their social security numbers. The table *Sales* is a huge table that contains information on the store's merchandise. The table *OldCustomers* contains the data of customers who used to buy from the store, but no longer do and had consequently deactivated their store credit card.

All employees who work in the sales section of the store have the same work responsibilities and are thus assigned to the same DB role. As part of their daily job activities, they access the table *Customers* to verify clients' identities. Each day, a few clients are expected to have their identities verified. The sales employees also access the table *Sales* to keep track of the merchandise. Since the table *Sales* contains important information that the organization cannot risk to lose, this table is backed up weekly. An automated process runs on behalf of the role of the sales employees every Friday at 9 p.m.

Bob who has been notified that he may be moved to another department is very upset about this possibility and thus decides to apply for positions at competing stores. Bob also decides to make a copy of the store's DB as he thinks that this data will be useful to him when he works at the new employer. Below are the steps that Bob could follow to access the data in each DB table. In each step, we explain how Bob's malicious accesses can be detected by our techniques.

- Bob may first send a query that selects all the rows in the table *Customers*. Although Bob has read permissions to *Customers* according to RBAC, the amount of data accessed by Bob's query significantly exceeds the amounts normally accessed by similar previous queries that reference *Customers*. We employ the techniques proposed in the previous chapter that capture the size of data that is normally accessed by the DB users in queries; therefore, the query by Bob will be flagged as anomalous.
- Similarly, all attempts by Bob to read from *OldCustomers* will be flagged as anomalous as this table has never been accessed by a user in Bob's role.
- In order to gain access to the table *Sales*, Bob will issue a query similar to the *Sales* table backup query at a time different from Friday 9 p.m. Since our techniques capture the consistent frequency of execution of each query, this query by Bob will be flagged as anomalous.
- Upon failing to send a query similar to the backup query at a time different from the scheduled backup time, Bob may try to synchronize his query with the backup query. The query sent by Bob will be flagged as anomalous, if it is issued after the legitimate periodic query. In case it is issued slightly before the periodic query, Bob will gain access to *Sales* and the actual legitimate query will later be flagged as anomalous; information on the anomaly and its reason is presented to the security administrator and Bob will be identified as part of the problem.

4.1.2 Architecture

The Profiler processes past logs of queries by the users of the T-DBMS offline during a training phase. The Profiler aggregates profiles of users who belong to the same role in order to form roles' profiles. The Profiler uses an SQL parser to parse the training queries and extract their syntactic features; in order to extract the semantic features of the training queries, the PB executes the training queries on a mirror of the T-DB. Therefore, either a snapshot of the T-DB at the start time of the training or a complete log of the T-DB is required.

To construct the roles' profiles, the Profiler processes the training log of each role in three stages as follows.

1. First, the Profiler detects the periodic queries in the log, their expected execution times and the relationships between the periodic queries, and records this information in the profile of the role under consideration.

2. The Profiler then removes the records related to the periodic queries from the log.

3. The Profiler then applies the training algorithms described in the previous chapter on the logs resulting from the previous stage. The goal is to infer the patterns of access of the users of the role to the T-DB's entities and the amounts of data that are retrieved from such entities in the training queries.

The detection phase starts after the processing of the logs of all the roles and the construction of the roles' profiles. During this phase, all connection and query requests sent by the users or applications to the T-DBMS are intercepted by an SQL proxy and relayed to the A-Detector. The A-Detector inspects the query requests and detects and logs mismatches between the queries and the profiles. Each input query is inspected in two stages:

1. The A-Detector, first, checks if the query is a periodic query that is expected from a user of the issuer's role at the time when the query is received by the proxy. If this is the case, the query is considered normal. Otherwise, the A-Detector proceeds to the next stage.

2. The A-Detector employs the role-based AD algorithms described in the previous chapter to determine if the input query *resembles* the queries in the training logs of the role of the issuer, which are *summarized* in the profile of this role. If the query mismatches the role's profile, the query is considered anomalous.

We assume that the proxy, the A-Detector and the T-DBMS servers reside in the same local area network (LAN) so that the expected arrival times of periodic queries that are recorded in the profiles can be compared to the actual arrival times of queries, which are recorded upon the receipt of the queries at the proxy.

Since one of our main design goals is to provide real-time AD, it is essential to prepare the result of inspection of a query before its result-set is shown to the issuer. We thus propose the execution of AD in parallel with the execution of select queries to provide low impact on their response times. Following this approach, the proxy is required to relay every input query to both the A-Detector and the T-DBMS. If the result of the execution of an input query is ready before the result of AD, the proxy does not relay the result-set records from the T-DBMS to the issuer until the response to the query is decided. Executing AD in parallel with the execution of queries that result in changes to the data in the T-DB is complicated because the changes made by a query that is considered anomalous by AD has to be rolled back if the response to the query is blocking the query.

4.2 Data Representation

In this section, we discuss the set of features of the DB queries that we consider in our approach. We consider two sets of features:

1. *Features that describe the individual SQL queries.* The features that represent an SQL query are encoded in a record that we refer to as the query's signature.
2. *Temporal features that describe several queries that have similar signatures.* These features are represented in the form of time series.

In the rest of this section, we describe the internal representation of each set of features.

4.2.1 SQL Queries Signatures

SQL queries are internally represented in the form of signatures. The structure of the signature of a query depends on the query type. In the following, we discuss the representation of two types of queries.

1. *Prepared statements and place-holders values generated by application programs.* Prepared statements are sent by data API's invoked by application programs, e.g., JDBC, to query DBMS servers. For a prepared statement to be executable,

Program

```
public class GetItemAverageRating extends Procedure {
    public final SQLStmt getAverageRating = new SQLStmt(
        "SELECT avg(rating) FROM review r WHERE r.i_id=?"
    );
    public void run(Connection conn, long iid) throws SQLException {
        PreparedStatement stmt = this.getPreparedStatement(conn,
                  getAverageRating);
        stmt.setLong(1, iid);
        ResultSet r= stmt.executeQuery();
        while (r.next()) {
            continue;
        }
        r.close();
    }
}
```

log

```
2017-05-02 19:08:05.482 EDT,"BEGIN","execute <unnamed>: BEGIN
2017-05-02 19:08:05.484 EDT,"SELECT","execute <unnamed>: SELECT avg(rating) FROM
review r WHERE r.i_id=$1","parameters: $1 = '23559'
2017-05-02 19:08:05.658 EDT,"COMMIT","execute S_1: COMMIT
```

(a) Executing a prepared statement and the generated log entries (the program is a code snippet extracted from the OLTP-Benchmark Epinions workload code)

Products (ID = 1001)

(0) p_ID	(1) p_name	(2) price
1	p1	1
2	p2	3
3	p3	5
4	p4	11
5	p5	13

Query

SELECT p_name, price
FROM Products
WHERE price < 10;

Quadruplet (c, l_T, l_A, s)
('SELECT', {1001}, {{1, 2}}, 'l_3')

(b) Example query on a T-DB table and the corresponding quadruplet

Figure 4.1.: Representation of SQL queries for the purpose of the detection of periodic queries

the application program issuing the statement provides values for the placeholders in the statement. The prepared statement portion of an executable SQL query contains information on the referenced tables, projected columns and columns referenced in the filter (where-clause) portion of the query that is responsible for characterizing the amount of data accessed by the query. Since this information is sufficient for profiling the query, prepared statements are used by our methods to represent this query type.

Prepared statements can be extracted directly from the training logs where they are recorded followed by the values of place-holders. Figure 4.1a shows an example Java program that uses JDBC to communicate with a PostgreSQL DBMS. The log entries generated as a result of calling the *run* method in the program with the second parameter (*iid*) set to 23559 are shown in the same figure. Place-holders are represented as $\$i$ in log entries, where i is a number that represents the index of the place-holder in the prepared statement. During the detection phase, the prepared statement of an SQL query and the associated place-holders values are sent to the JDBC module of the T-DBMS and can be captured by the proxy accordingly.

2. *Exact query strings.* An exact query string received by the T-DBMS is either issued by a user connected to the T-DBMS through a user interface (UI) tool, e.g., the DB terminal or a graphical UI (GUI) tool, such as the pgAdmin tool used to access PostgreSQL, or by an application program in which queries are encoded as constant strings or composed based on other strings and user inputs.

   Queries of this type are internally encoded in the form of quadruplets. A quadruplet QD that represents an SQL query Q contains information on the tables referenced in Q, the columns projected in Q and the selectivity of Q's result-set; QD has the form: (c, l_T, l_A, s), where c represents the command type of Q, l_T is the list of ID's of the tables referenced in Q, l_A is the list of attributes projected in Q, i.e., the attributes whose values appear in Q's result-set, and s represents the selectivity level of Q. We consider four levels of selectivity: l_1, l_2, l_3, and l_4, which represent the ranges [0, 0.25[, [0.25, 0.50[, [0.50, 0.75[, and [0.75, 1.00], respectively. Figure 4.1b shows an example query that accesses the DB table *Products* shown in the same figure and the quadruplet representation of the query.

   The selectivity of a query is computed based on the SQL command type of the query as follows. For a select query that references a single table, the selectivity of the query is equal to the result of dividing the (actual or expected) number of rows in the query's result-set by the number of rows stored in the referenced table. For a join select query that references more than one table, the selectivity is equal to the result of dividing the (actual or expected) number of rows in the query's result-set by the number of rows in the Cartesian product of the tables referenced in the query. Other types of queries (update, insert and delete types) access only one table; the selectivity of a query that is one of these types is equal to the number of rows *manipulated* by the query divided by the number of rows of the referenced table.

According to the quadruplet representation, different queries can be encoded similarly. We consider this choice adequate in most cases. A possible extension to this representation would be to add information on the predicates of the where-clause of the query. These predicates are in the form: tk_1 *op* tk_2, where *op* is an operator, and tk_1 and tk_2 can be either references to the attributes of the tables accessed in the query, expressions that reference the tables' attributes, or constants. In general, storing the constants in the where-clause in the query representation is incorrect; two queries should be considered similar, if they are similar in all aspects, but have different values of constants in the where-clause. Another alternative is to store information on the attributes referenced in the where-clause in the representation; however, this leads to a very strict representation of manually-issued queries that the users choose from a large space of valid queries.

It can be noted that prepared statements can be encoded in the form of quadruplets. Each form of representation has its pros and cons. The quadruplet representation of a query provides data for analysis, since it encodes information on the results of parsing and planning the query. On the other hand, the prepared statement representation can be inferred directly from the query request and thus does not require parsing or planning the query. However, the string of the prepared statement includes information on the code location at which the query is executed. Two queries that have been issued from different code locations and have subtle differences in their prepared statements' strings may have similar quadruplet representations, but can be differentiated using their prepared statements representations.

4.2.2 Time Series Representation

The distribution of the time-stamps of execution of one query or a group of queries that have the same signature by the users of a role during a specific time interval is represented in the form of two time series:

1. *Real time series (RTS).* An RTS is a typical time series. The length of an RTS, in terms of the number of time buckets, is equal to the length of the time interval represented by the series divided by the time resolution (R). The number of time-stamps that fall within the time interval represented by each bucket are associated with the bucket and stored in the RTS.

2. *Hypothetical time series (HTS).* An HTS is similar to an RTS in all aspects except that the time scale of the HTS is not real; in the HTS representation, we assume that each month contains 31 days.

Figure 4.2(a) shows example time intervals and information on the RTS and HTS representations of queries executed during these intervals for different values of R. Figure 4.2(b) shows an example log file and the RTS representation of the log entries; the HTS representation of the log is similar to the RTS representation, since the log covers a short time interval.

4.3 Training Phase

The purpose of training is to detect the periodicity of the queries in past logs of users' accesses and the relationships between periodic queries that are issued together. For this purpose, the Profiler processes the queries executed by the users of each role by first converting each query into a signature and then aggregating the queries' time-stamps according to their signatures. The Profiler constructs the RTS and HTS representations of each signature's time-stamps disregarding the signatures that have very few associated time-stamps.

In order to detect the periodicity in a signature's time series, the Profiler applies the following steps.

1. Execute Algorithm Period-Detect to find candidate lengths of the period in the time series of the signature. Ideally, a general time series may have more than

Time Interval	***R***	**RTS count**	**HTS count**
2017-1-1 08:00:00:000 to 2017-1-1 12:00:00:000	1 hour	4	4
2017-1-1 08:00:00:000 to 2017-5-1 12:00:00:000	1 day	121	125

(a) Time intervals and the bucket count of the corresponding time series

Query	**Time-stamps**
SELECT * FROM Products WHERE price < 10;	2017, 1, 1 1 : 04 : 01 : 011 2017, 1, 1 1 : 04 : 51 : 101 2017, 1, 1 3 : 30 : 42 : 420 2017, 1, 1 4 : 40 : 21 : 833

RTS	**Interval Start**
{	
2,	2017, 1, 1 1 : 04 : 01 : 011
0,	2017, 1, 1 2 : 04 : 01 : 011
1,	2017, 1, 1 3 : 04 : 01 : 011
1,	2017, 1, 1 4 : 04 : 01 : 011
0	2017, 1, 1 5 : 04 : 01 : 011
}	

(b) Example log file and the corresponding RTS representation ($R = 1$ hour)

Figure 4.2.: Time series representation of query execution time-stamps

one period. However, we consider a special type of time series that represent *important* queries, e.g., backup queries that access large amounts of data. We assume that a query of this type may have only one period. The rest of the discussion follows this assumption, but the described approach can be applied to time series that have more than one period.

2. Execute Algorithm Period-Filter on the periods detected in the first step to eliminate false-positives and to find the time intervals when the periodic queries are expected to be received at the T-DBMS.

3. Detect the relationships between periodic queries and the order in which related queries should be expected.

We discuss each of the previous steps in details in the rest of this section.

4.3.1 Periodicity Detection

The Profiler applies Algorithm Period-Detect in order to find candidate periods in the RTS and HTS of the time-stamps associated with a signature under consideration.

The purpose of analyzing both the RTS and the HTS is to differentiate between **absolute periods** and **calendar-based periods**; the two period types are different with respect to the approach used for describing the lengths of each type. An absolute period is represented as an absolute time interval, i.e., number of seconds, and can thus be detected using the RTS representation. Calendar-based periods occur every month or every few months. Calendar-based periods can be detected using the HTS representation, since this representation considers that all months have equal lengths.

Algorithm Period-Detect Algorithm Period-Detect follows the standard approach for periodicity detection [30] that relies on autocorrelation. The input to Algorithm Period-Detect is a time series and the output is a set of candidate lengths of the period existing in the input series. The algorithm operates in two main steps:

1. *Computing the autocorrelation of the input series.* The autocorrelation of a time series is the correlation of the series with its shifted copy as a function of the amount of shift (referred to as lag). For a discrete time series $S = \{s_1, s_2, ..., s_n\}$, an estimate of the autocorrelation of S is obtained as:

$$\widehat{A}(k) = \frac{1}{(n-k)\sigma^2} \sum_{t=1}^{n-k} (s_t - \mu)(s_{t-k} - \mu)$$

   where μ and σ^2 are the mean and variance of the sequence, respectively, and k is the length of the lag.

2. *Finding the lags that correspond to peak values in the autocorrelation.* Since the value of the autocorrelation at a specific lag k is a measure of the similarity of the series to the result of shifting the series' entries by the value of k, the value of the autocorrelation is expected to be high for values of k equal to the lengths of the periods in the series; obviously, the maximum value of the autocorrelation occurs at $k = 0$. Algorithm Period-Detect thus considers the non-zero lags that correspond to the top N autocorrelation values as candidate periods of the input series.

4.3.2 Eliminating False-Positives

The output of Algorithm Period-Detect is a set of N values that represent the candidate lengths for the period in the input time series, if one exists. Out of the N values, $N - 1$ values are false-positives, if the input series is periodic. If the input series is aperiodic, all N values are false-positives.

In order to eliminate the false-positives, we developed Algorithm Period-Filter that detects the periodic entries in a time series given the candidate lengths of the period in the series. Besides eliminating false-positives, Algorithm Period-Filter has the following benefits.

1. *Estimating the time interval during which the periodic queries are expected to arrive at the T-DBMS.* This is important for taking into account the effect of the network latency on the arrival times of the periodic queries and differentiating between **manual and program-generated (automated) periodic queries**; the main difference between the two types is that the time of an automated periodic query is more specific than the time of a manual periodic query. We consider the difference between the two types by recording in data-structures an interval for each periodic query during which the query is expected to be received at the T-DBMS. Manual periodic queries have longer intervals of expectation and can be differentiated from automated periodic queries accordingly.

2. *Estimating the ratio of the periodic queries that can be missed.* This is useful for differentiating between stopped periodicities and periodic queries that are not issued in some occasions, e.g., corporate vacations. Detecting stopped periodicities is essential for the maintenance of accurate profiles that are helpful in the proper detection of anomalies.

Algorithm Period-Filter. The inputs to Algorithm Period-Filter are a time series and the candidate lengths of the period in the series. Algorithm Period-Filter checks each candidate length by considering a window at the start of the series whose length is equal to the candidate period length; starting at each access in this window, the algorithm checks if there is an access that is repeated every period of the input length in the series. The result of one round of search is the number of missed accesses as a percentage of the total number of expected accesses, the time of the first periodic access in the series, and the interval around the estimated time of the periodic access

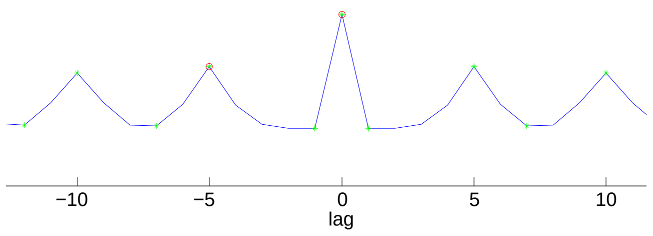


Figure 4.3.: Result of computing the autocorrelation for a periodic time series with period $p = 5$

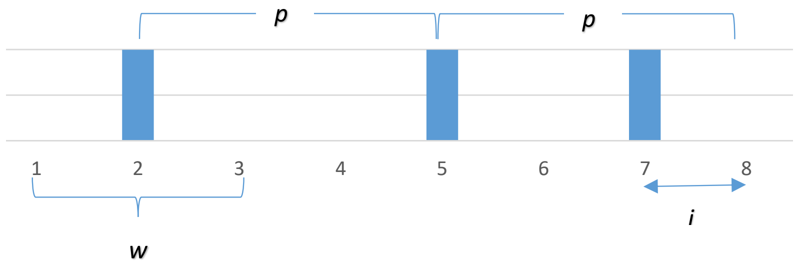
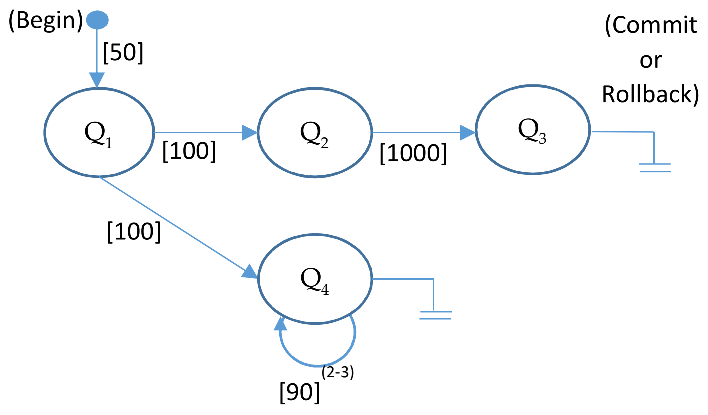


Figure 4.4.: Algorithm Period-Filter search example ($w = p = 3$, $i = 1$)

when later accesses are expected to arrive at the T-DBMS. A candidate length is accepted as a period of the series, if the percentage of missed accesses of this period is less than a predefined threshold value (p_{thr}).

Algorithm Period-Detect often generates false-positives that are multiples of the actual period length. For example, consider the autocorrelation of an input series that has a period of length equal to five units shown in Figure 4.3. The peaks in the autocorrelation occur at ..., -10, -5, 0, 5, 10, ... lags. Algorithm Period-Filter accounts for this case by considering the absolute values of the output lags of Algorithm Period-Detect, sorting these values and eliminating duplicates before checking the candidate lengths. Algorithm Period-Filter checks if a candidate length is a multiple (or close to a multiple) of a value confirmed earlier by the algorithm as the length of the period in the input series; in this case, the new candidate length is considered a false-positive of Algorithm Period-Detect.

Figure 4.4 shows the steps followed by Algorithm Period-Filter for a candidate period length equal to three buckets ($p = 3$). The search starts from the entries in the window w whose length is equal to p. Considering the accesses that start at the second entry in the time series, the third expected access occurs one entry before the



```
Begin
    Execute $Q_1$;
    If <cond>
        $RS_2$ = Execute $Q_2$;
        For each row $r$ in $RS_2$
            Print $r$ to file
        Execute $Q_3$;
    Else
        While <cond>
            Execute $Q_4$;
End
```

Figure 4.5.: Example PAG

correct one. Algorithm Period-Filter can associate this access with the periodic access pattern as the search is extended to entries within $i = 1$ buckets from the expected location.

4.3.3 Inferring the Relationships between Periodic Queries

Automated periodic DB accesses that are issued by application programs may contain more than one query. The time of each query depends on many factors, e.g., the structure of the program that generates the queries and the system on which the program runs. However, specifying the time of the first query in the generating program is more feasible than specifying the time of subsequent ones, since the execution times of the subsequent queries depend on more factors than the execution time of the first query, e.g, the sizes of the result-sets of previous queries for programs that scan and save queries' result-sets. The expected sequence of query submission can be inferred by profiling application programs [17, 18] or by mining query logs. We chose to follow the later approach in this paper.

Besides the proper tracking of queries, recording the sequence of submission of related periodic queries is useful for detecting program failures and query anticipation. The automatic recovery from failures by sending some related periodic queries may require the execution of the whole periodic sequence; in this case, the A-Detector has to take into account this scenario and allow the re-execution of some periodic queries. Planning the execution of queries that are expected in a sequence is also useful for reducing the time of the preparation of the response to these queries.

For the purpose of inferring the expected sequence of periodic queries, the Profiler organizes the log queries into sessions and aggregates sessions according to the signatures of the first queries in the sessions; for each signature, the Profiler constructs and analyzes time series of the start times of the sessions related to the signature under consideration in order to find information on the period in the series, if one exists. The Profiler then organizes sessions of the start signatures that are part of periodic accesses in the form of graphs of a special structure referred to as periodic accesses graph (PAG).

In a PAG, nodes represent queries; an edge (e_{ij}) that connects the nodes of queries q_i and q_j is present in the graph, if q_j appeared after q_i in one or more training sessions. The weight of e_{ij} stores the maximum length of the time interval between q_i and q_j according to the training logs.

Figure 4.5 shows an example PAG for the periodic program shown in the same figure. q_1 is the first query in the periodic code block and either q_2 or q_4 has to follow q_1. q_3 is the successor of q_2. Since the result-set of q_2 is scanned and saved by the program, the time interval between q_2 and q_3 is relatively longer than the time intervals between other queries. According to the training log, q_4 is expected to be repeated twice or thrice.

4.3.4 Format of Profiles

Periodic accesses are internally represented as PAG data-structures. All PAG structures are stored in a hash-table that we refer to as H_{pag}. The key to a PAG structure in H_{pag} is the signature of the first query in the periodic access that the structure represents. Nodes in a PAG are stored as PAG_node structures. A PAG_node n that belongs to a PAG *pag* has the form: $(q_s,\ t_p,\ n_r,\ l_{\text{pags}})$. $n.t_p$ is the maximum allowed time between the start of the session and the arrival of $n.q_s$, if n represents the first node in *pag*; otherwise, $n.t_p$ represents the maximum allowed time between the arrival of $n.q_s$ and its predecessor. $n.n_r$ is either zero or a positive integer; a zero value indicates that $n.q_s$ cannot be repeated in an execution of *pag*; while a positive value indicates that $n.q_s$ can be repeated. $n.l_{\text{pags}}$ is a list of pointers to the PAG_node structures of the successors of n.

A PAG *pag* is internally represented in the form: $(q_s,\ t_e,\ [i_{\text{start}},\ i_{\text{end}}],\ p_m,\ p_f)$, where q_s is the signature of the first query in *pag*, t_e is the time of the next expected execution of *pag*, i_{start} and i_{end} are the start and end of the interval around t_e (and later expected execution times) when the periodic access can be accepted by the proxy, p_m is the percentage of executions of *pag* that can be missed, and p_f is a pointer to the first PAG_node in *pag*.

4.4 Detection Phase

During the detection phase, the A-Detector keeps track of the mapping between each connected user and his/her activated role. The A-Detector also associates each user session that is currently executing a periodic access with an instance of a PAG_node; this instance represents the current state of the session in the PAG structure of the periodic access that the session is currently executing.

Given an input query to inspect, the A-Detector differentiates between the following three cases.

1. *First query in a session.* In this case, the A-Detector checks if the input query is part of a periodic access by searching for the query's signature in H_{pag}. If an entry *pag* is found, the A-Detector computes the interval of time when the periodic access should be expected at the T-DBMS as $[pag.t_e{+}pag.i_{\text{start}},\ pag.t_e{+}\ pag.i_{\text{end}}]$. If the time of the start of the session lies within the expected interval, the A-Detector checks if the time between the session start and the query arrival is less than $pag.p_f.t_p$ in order to consider the input query part of the periodic access. In this case, the A-Detector associates $pag.p_f$ with the session, stores

information on the next expected states in the session by referring to $pag.p_f.l_{\text{pags}}$ and considers the query normal.

2. *Query in a session that is currently executing a periodic access.* In this case, the A-Detector searches the next expected states associated with the session of the input query for a node that stores the same signature as the input query. If no node is found or the next expected state of the session is null, the input query and all subsequent queries in the session are considered anomalous. Otherwise, if a node n is found, the A-Detector checks if the time between this query and the previous query in the same session is less than $n.t_p$. If this condition is violated, the query is considered anomalous. Otherwise, the query is considered normal and the A-Detector stores the next expected states of the session by referring to $n.l_{\text{pags}}$; if the input query can be repeated ($n.n_r > 0$), n is added to the next expected states of the session.

3. *Session end.* When a session that is currently executing a periodic access ends, the A-Detector checks if the next expected state in the session's PAG is null. If this is not the case, the incidence is recorded in the log as an anomaly.

4.5 Experimental Evaluation

We now discuss our approach for the experimental evaluation of the techniques proposed in this chapter and the results of the evaluation. The goal of the experiments is to assess the accuracy of the queries' periodicity information stored in training profiles and the accuracy of matching the detection queries to the profiles.

In all experiments, we relied upon the OLTP-Benchmark [31] workloads for generating the datasets used for the evaluation. The OLTP-Benchmark contains different workloads. Each workload operates on a pre-designed DB that is loaded with data generated by the benchmark's programs. A workload contains one or more scenarios and a scenario contains one or more queries. The workload distribution across the different scenarios, the time for running the workload and the rate of query submission are provided by the benchmark's user; the benchmark workload can thus be configured for the purpose of generating data that applies for different testing and experimentation scenarios.

In order to generate the datasets used for the experiments, we ran several periodic scenarios in the OLTP-Benchmark. These scenarios are mainly part of the Epinions, TATP, and TPCC workloads. We ran the periodic scenarios in a client-server architecture where the benchmark's query-generation programs are run on a client machine that sends queries to a PostgreSQL DBMS server. Both the client and server machines reside in the same LAN. This architecture is useful for taking into account the network latency and thus recording and tracking realistic intervals for the arrival of periodic queries at the T-DBMS.

The experiments are divided into three sets, which have different purposes:

1. *Set A - Experiments for measuring the accuracy of the detection of periodicity in training time series.* This set of experiments measures the accuracy of Algorithms Period-Detect and Period-Filter described in Section 4.3 for the detection and characterization of periods in time series of queries' signatures. We evaluated the performance of the algorithms on five different scenarios of the Epinions workload. We chose the period length to be one second and ran the algorithms for six different values of R: 2, 10, 50, 100, 200, and 500 milliseconds. This results in a total of 30 cases to check. We set the parameter N in Algorithm Period-Detect to 30 and the parameters p_{thr} and i in Algorithm Period-Filter to 20% and 500 milliseconds, respectively. We chose i to be a small value because the periodic queries that appear in the evaluation datasets are program-generated and the DB clients and proxy machines are close in terms of network proximity.

   The result indicates that the periods lengths and the intervals of expected arrival times of the periodic queries are correctly detected in all cases except for three cases; in the failing cases, the algorithms result in many false-positives that are multiples of the correct periods' lengths. Meanwhile, the actual periods are not detected by the algorithms and are thus considered false-negatives. Algorithm Period-Filter can be updated to take into account the failing cases by adding a final step to the algorithm as follows. The algorithm checks if the output contains many periods that are (close to) multiples of a specific value; in this case, this value is fed back to the start of algorithm, i.e., checked to find if it is an actual period of the input series.

2. *Set B - Experiments for measuring the accuracy of the detection of the first queries in periodic accesses.* The purpose of this set of experiments is to measure the accuracy of matching queries to the periodicity information stored in profiles. To generate the training and test queries, we ran the same scenarios used in the experiments of Set A for an interval of length equal to 100 time units setting the query rate to one query/unit; this results in about 100 generated queries in 100 seconds. We used 80% of these queries for training and the remaining 20% for measuring the matching accuracy. The length of the training log is considered equivalent to the length of a one-year log of weekly periodic queries. For the purpose of constructing the time series used for training, we set R equal to 500 milliseconds. The result indicates that 5% of the test queries fall outside the expected intervals and are thus considered false-positives.

3. *Set C - Experiments for measuring the accuracy of matching session queries to PAGs.* We measured the matching accuracy for scenarios of the Epinions and TATP workloads of the OLTP-Benchmark, which contain a few non-repeating queries. We also measured the matching accuracy on more complex PAGs that contain many queries; some of them are repeated and thus result in PAGs that contain loops. The complex PAGs are the result of running some scenarios of the TPCC dataset. Each training log that belongs to a specific scenario contains about 80 periodic code blocks; these constitute 80% of the generated sequences. The remaining 20% is used for testing. The result indicates 100% matching accuracy for programs that contain 2$\sim$3 queries. The error is less than 5.6% for complex programs that contain repeating states. Other simpler PAGs have 100% matching accuracy.

4.6 Conclusions

In this chapter, we presented techniques for the detection of periodic queries in DB logs. Our techniques are able to flag unexpected changes in the frequencies of execution of periodic queries.

5 DETECTION OF ANOMALOUS SEQUENCES OF QUERIES THAT RETRIEVE DATASETS LARGER THAN NORMAL SIZES

In this chapter, we introduce AD techniques for the detection of scenarios of aggregation of data retrieved from the T-DB. A malicious insider may choose to aggregate the data returned by multiple queries rather than execute a single query that retrieves all the target data in order to hide his/her tracks of unnecessary data retrieval. The detection of this misuse scenario requires temporal monitoring of user actions in order to infer correlated actions indicating unusual data aggregation activities.

Our techniques are of two types: online and offline. Online AD is used as a means of real-time detection of abnormal queries that access excessive data amounts. Online AD relies mainly on comparing the amount of data retrieved from DB relations with threshold levels computed based on past logs of users' accesses. The inspection of individual queries requires parsing and analyzing these queries to extract their syntactic and semantic features. Specifically, the features considered for each query are the relations referenced in the query and the selectivity of its result-set. Online AD is meant to be fast in order to provide the result of the inspection of queries before their result-sets are returned to the issuers.

On the other hand, offline AD captures and tracks features of the user queries that are not considered by online AD and thus performs more thorough inspection of the users' behavior. The purpose of offline AD is two-fold: (1) the detection of anomalous query sequences and (2) the detection of anomalous users' sessions of connection to the T-DBMS. The method for inspecting query sequences is similar to the one for inspecting individual queries, except that the former method requires the execution of some queries on the monitored DBMS to account for the case when the query analyzer fails to provide accurate estimates of the selectivities. The inspection of users' sessions is performed using machine learning techniques, such as novelty detection and unsupervised learning. As opposed to the existing AD approaches, neither offline nor online AD assume clean (anomaly-free) training data. Existing anomalies in the logs used for building the AD models are automatically detected and disregarded.

Distinguishing false-positives from true-positives in the presence of uncertainty is particularly challenging as false-positives could harm the reputation of individuals when rigorous response actions are taken against detected malicious insiders [5]. Therefore, we report a degree of risk associated with each anomalous query and sequence of queries. We also define distance metrics to quantify the differences between the anomalous DB sessions and the normal ones. These values can be used by an automated system to respond to queries and take actions against users deemed to be malicious.

Due to the scarcity of real intrusions data for ground truth test and evaluation [32], we describe our approach for the generation of data and queries that mimic real data misuse scenarios. The data we generate are based on the workloads of the OLTP-Benchmark [31]. We used the generated queries in the evaluation of our techniques. The result of the evaluation indicates that our techniques are very effective.

The remainder of this chapter is organized as follows. We discuss the architecture in which the proposed techniques can be deployed in Section 5.1. In Sections 5.2 and 5.3, we present detailed algorithms for the proposed techniques. We describe our approach for the experimental evaluation of the proposed techniques and discuss the results of the evaluation in Section 5.4. Section 5.5 concludes the chapter and discusses future work.

5.1 Architecture

Our techniques operate in two successive phases. The first phase is a training phase during which a Profiler module is fed with past logs of the T-DBMS users' accesses. The job of the Profiler is to capture the T-DB users' access patterns based on the logs and construct the profiles of these users accordingly. The Profiler thus combines individual users' profiles to form roles' profiles, which are smaller in number and can thus be better managed.

The detection phase, which starts after the training is complete, is the second phase. During this phase, all role-activation, connection, disconnection and data querying requests issued to the T-DBMS are intercepted by a query interceptor (also referred to as SQL proxy) and relayed to an anomaly detector (A-Detector) module, whose job is to monitor connections and inspect users' queries.

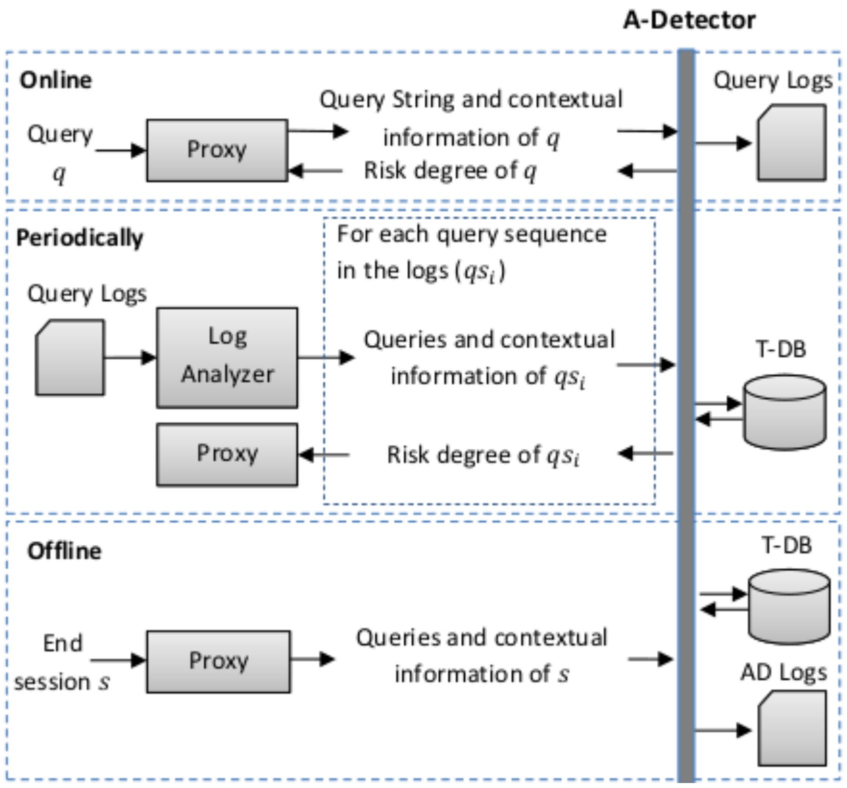

A-Detector
Online
Query q
Proxy
Query String and contextual information of q
Risk degree of q
Query Logs
Periodically
Query Logs
Log Analyzer
For each query sequence in the logs (qs_i)
Queries and contextual information of qs_i
Proxy
Risk degree of qs_i
T-DB
Offline
End session s
Proxy
Queries and contextual information of s
T-DB
AD Logs

Figure 5.1.: Timing of the application of the AD techniques used for the detection of temporal data ex-filtration

Since data ex-filtration is performed by first extracting the data of interest from the T-DB and then using other methods to export the data outside the system, our work focuses on the inspection of data retrieval queries, i.e., select queries. Role activation and connection requests are observed by the A-Detector to set up the internal data-structures used to track the users' behavior and detect anomalies.

For the purpose of query and session inspection, the A-Detector employs a DB parser and analyzer (optimizer), which are different from the DB components of the T-DBMS. The purpose is to extract the queries' syntactic and semantic features that characterize the relations referenced in queries and the sizes of the queries' result-sets. Since the optimizer fails, for some queries, to provide accurate estimates of the sizes of result-sets, the A-Detector needs to communicate with the T-DBMS to extract the information required for the proper inspection of these queries.

The AD techniques we propose operate at three stages during the users' sessions with the T-DBMS: (1) before the execution of each user query, (2) after the execution of each user query or group of queries, and (3) after the end of each user session. The result of the first stage is sent to the proxy to decide on the response to the input queries in real-time in addition to being recorded in the logs. Figure 5.1 illustrates the three AD stages. The first two stages are described in Section 5.2 and the third stage is described in Section 5.3.

5.2 Session Tracking

In this section, we present an approach for tracking the sizes of data retrieved by queries and sequences of queries during sessions of the users' connection to the T-DBMS. We refer to this approach as session tracking (ST).

5.2.1 Detection of Anomalous Queries

ST records in the profile of each role information on the sets of relations that are referenced together in training queries executed by the role's users. ST also records a selectivity threshold associated with each set indicating the maximum selectivity according to which a select query issued by the role's users typically accesses from the DB relations.

Given an input select query to inspect, the A-Detector checks the query upon the profile of the role of the query's issuer. The query is considered anomalous in two cases:

1. If it references a set of relations that does not have a corresponding record in the profile of the role of the issuer.

2. If a record that corresponds to the referenced relations is located in the profile of the issuer's role, but the selectivity of the query is higher than the threshold stored in the record.

The A-Detector reports each anomalous query associated with a risk degree, which characterizes how different the query is from the training profile of the role of the issuer. An anomalous query that matches the first criterion described above has a 100% risk degree, whereas the degree of risk of a query that matches the second criterion is computed based on the threshold level associated with the located record (s_{ind}) and the selectivity of the query (s_{act}) as:

$$\min(100, \frac{s_{\text{act}} - s_{\text{ind}}}{s_{\text{ind}}}\%).$$

This information allows the administrator and a configurable automated response system to decide on the response to the anomalous query.

The selectivity of a query is computed as follows. If the query references a single relation, the query's selectivity is the result of the division between the number of rows in the query's result-set and the number of rows stored in the referenced relation. The selectivity of a join select query is computed as the result of the division between the number of rows in the query's result-set and the number of rows in the Cartesian product of the relations referenced in the query.

The Profiler determines the number of rows in the result-set of a query used for training the AD system by executing the query on a copy of the T-DB when the query is issued based on the training log, and observing the result-set. During the detection phase, the A-Detector employs the optimizer to compute an estimation of the result-set size of the query under inspection.

Employing the optimizer for result-set size estimation is more adequate than query execution for a real-time AD system, since query planning can be performed at the A-Detector in parallel with the query execution at the T-DBMS. This approach thus reduces the time required for query inspection that is considered part of the time of the response to the query. The approach also reduces the overhead on the T-DBMS and increases its throughput, since no additional queries are to be executed at the T-DBMS for the purpose of query inspection. However, this approach adds the requirement of the availability of the T-DB or statistics on the T-DB data to the optimizer component of the A-Detector.

The challenges associated with using the optimizer for result-set size estimation have been discussed and addressed by our previous work [9, 10, 33]. Our previous approach relies on computing per-table selectivity levels and uses machine learning for matching queries under inspection to training queries. Although such selectivity levels are useful for capturing detailed information on queries, the computation of these values involves many heuristics and approximations. The approach described in this paper avoids these approximations by capturing selectivities of queries instead, but does not consider detailed information on the structure of valid queries, e.g., the columns that are projected in the queries' result-sets; encoding such information in the

Table 5.1.: Example DB and sessions' logs

(a) Products

ID	Price	c_ID
1	1	1
2	3	1
3	10	1
4	20	2
5	50	2
6	60	3

(b) Categories

ID	Descr
1	Personal Care
2	Beauty
3	Health

(c) Query log

Session 1
1. SELECT * FROM Products WHERE price <= 10
2. SELECT * FROM Products WHERE price >= 5 AND price <= 20
3. SELECT p.name, p.price FROM Products p, Categories c WHERE p.c_ID = c.ID AND c.descr = 'Personal Care'
Session 2
1. SELECT p.name, p.price FROM Products p, Categories c WHERE p.c_ID = c.ID AND c.descr = 'Health'

profiles would lead to a very strict selection of valid queries. The approach described in this paper has the important advantage that it can be easily extended to support temporal data misuse detection as discussed next.

5.2.2 Detection of Anomalous Query Sequences

A query sequence (qs) is a group of queries that have the following characteristics.

1. All queries in qs are of select type, refer to the same set of DB relations, and have been executed by a specific DB user during one session.

Table 5.2.: Example ST profile

$\boldsymbol{R}$	$s_{\textbf{ind}}$	$s_{\textbf{agg}}$	P_S
Products	$\frac{1}{2}$	$\frac{2}{3}$	1: {price <= 10, price >= 5 AND price <= 20}

Products,Categories	$\frac{3}{18}$	$\frac{3}{18}$	1: {Products.c_ID = Categories.ID AND Categories.desc = ‘Personal Care’}
			2: {Products.c_ID = Categories.ID AND Categories.desc = ‘Health’}

2. Each query in qs is considered normal, when inspected individually by ST.

ST inspects sequences of queries to detect anomalous ones. A query sequence is considered anomalous, if the total (aggregate) selectivity of the data retrieved by the queries in the sequence exceeds a threshold level captured during the training phase. Given that each query is inspected individually by the A-Detector and that the proxy takes the necessary response based on the AD result, the inspection of query sequences does not have to be performed on a per-query basis and can thus be performed either periodically or after the execution of a number of queries.

A record in the profile of a role, which stores information on a set of relations referenced by the users of this role in one or more training queries and the threshold on the selectivities of single queries, also stores a threshold on the aggregate selectivities of sequences of queries that reference the set in one session.

To compute the threshold level for a set of relations R, the Profiler finds the sequences that refer to R in the training logs, executes the queries in each sequence and combines the result-sets of queries that belong to the same sequence removing duplicate rows. The threshold level is computed as the maximum value among the selectivities of the sequences.

Algorithm 5.1: Training Phase of ST

```
Input: training logs of a role $rl$.
Output: profile of $rl$ ($H_{rl}$).
1. Parse the training queries to populate the entries in $rl$'s profile and the predi-
   cates in the predicates' list of each entry in $H_{rl}$, i.e., $P_S$.

2. Compute $s_{agg}$ for each record $rc$ in $H_{rl}$ as follows.
```

```
2.1. For each session $s$ represented by an entry in $rc.P_S$, execute a query of
     the form:

     $Q_{\text{train}}$: SELECT COUNT(*) FROM $< R >$ WHERE $< P >$,

     Where $< R >$ is replaced with the result of the comma-separated con-
     catenation of the names of the relations in $rc.R$ and $< P >$ is replaced
     with a string composed by combining all the predicates associated with $s$
     in $rc.P_S$ using OR operators.
2.2. Compute the list of selectivities of the queries executed in the previous
     step ($S_S$).
2.3. Apply Grubb's test [34] to detect the outliers in $S_S$ and compose a new
     list of selectivities $S'_S$ after removing these outliers from $S_S$.
2.4. Set the value of $rc.s_{\text{agg}}$ to the maximum value in $S'_S$.
```

Table 5.2 shows an example profile that is based on the query log shown in Table 5.1(c) and references the DB relations in Tables 5.1(a) and 5.1(b). Each record in the profile of a role is a quadruplet of the form: $(R, s_{\text{ind}}, s_{\text{agg}}, P_S)$, where s_{ind} is the individual selectivity threshold and represents the maximum selectivity that a query that references R can reach, s_{agg} is the aggregate selectivity threshold and represents the maximum selectivity that the result-sets of query sequences that reference R can reach in one session, and P_S is the sessions' predicates list and is of array type. The predicates in P_S are used for training purposes only. However, keeping this information in the profile is useful because it avoids re-parsing the training queries for the purpose of updating the profiles. Profile updates are necessary when the data stored in the DB changes significantly.

Algorithm 5.1 shows the steps followed by ST for the construction of the profile of a role. A few remarks must be made on the algorithm.

- Rather than adding the cardinalities of the result-sets of queries in a query sequence in order to find its aggregate selectivity, the algorithm executes the query Q_{train} that combines the result-sets of the individual queries and excludes repeated rows, and thus produces accurate results of the aggregate selectivities.

- Since the training data may contain previous attempts for data ex-filtration, removing the outliers from the training data performed in Step 2.3 is essential for capturing normal and accurate aggregate selectivities.

During the detection phase, the A-Detector stores the state of each user session in an in-memory session variable of hash-table type. The hash-table associated with a session s is referred to as H_s. Each record in the profile of the role of the owner of s has a corresponding record in H_s. A record in H_s that represents a set of relations R is a quiplet of the form: (R, $s_{\text{ind}}, s_{\text{agg}}, s_{\text{curr}}, P_s$), where s_{ind} and s_{agg} are copied from the corresponding attributes in R's record in the profile of the role of the session's owner, and s_{curr} is the current aggregate selectivity of R and represents the actual or an estimate of the actual aggregate selectivity of the query sequence that accesses R and belongs to s; the list of predicates of these queries is stored in P_s.

Algorithm 5.2 shows the steps followed by the A-Detector to analyze the effect of an input query q executed during a user session s on the aggregate selectivity of the query sequence that includes q. The A-Detector tries to avoid the execution of queries by employing the optimizer for estimating the selectivities of queries and keeping track of the aggregate selectivities of sequences (Steps 2.3 and 2.4). However, if the aggregate selectivity of a sequence exceeds the threshold, the A-Detector cannot make an immediate decision, since the value of the current selectivity (s_{curr}) is only an estimate and repeated rows in the result-set of the sequence are not excluded in the calculations. In this case, the A-Detector executes the query Q_{train} against the T-DB to compute the actual value of s_{curr} and make a decision on the sequence of q (Step 2.2).

Algorithm 5.2: Detection Phase of ST

Input: a select query (q) executed during a user session s.

Output: whether the query sequence that includes q is anomalous or normal.

1. Parse q to extract its range-tables (R). Compose the key (k) that represents R and use k to search H_s for a record that corresponds to R. If no record is found, consider the sequence of queries executed during s that reference R anomalous, and terminate.

2. If a record rc that represents the set of relations R is located in H_s, find if the threshold of access to R has been exceeded during s as follows.

   2.1. If $rc.s_{\text{curr}} \leq rc.s_{\text{agg}}$, consider the query sequence that reference R during s normal.

2.2. If $rc.s_{\text{curr}} > rc.s_{\text{agg}}$, $rc.s_{\text{curr}}$ could be an estimate of the selectivity of the accessed portion of R and the AD result cannot be inferred immediately. In this case, compute the accurate value of $rc.s_{\text{curr}}$ by executing a query similar to Q_{train} replacing $< P >$ with the string composed by combining the predicates in $rc.P_s$ using OR operators, and compare the values of $rc.s_{\text{curr}}$ and $rc.s_{\text{agg}}$. If $rc.s_{\text{curr}} > rc.s_{\text{agg}}$, consider the sequence of queries that reference R anomalous and compute the anomaly degree of the sequence as: $\min(100, \frac{rc.s_{\text{curr}} - rc.s_{\text{agg}}}{rc.s_{\text{agg}}}\%)$. Otherwise, consider the query sequence normal.

2.3. Employ the optimizer to estimate the selectivity of q (sl).

2.4. Update rc by adding sl to $rc.s_{\text{curr}}$ and appending the predicate of q to $rc.P_s$.

5.3 Session Evaluation

5.3.1 Basic Approach

One drawback of ST is that it does not take into account the correlations between the values of the aggregate selectivities of queries executed in the same sessions. As an example, consider, hypothetically, an employee named “Bill” who works for a corporate. In order to perform his job responsibilities, Bill has privileges to access the corporate’s DB. The clients of the corporate belong to two main categories and thus their records are stored in two different DB relations. Since Bill has a maximum capacity for processing the clients’ records, it is possible that in one session Bill retrieves many records of one relation, many records of the other relation, or a few records from each relation. Due to the use of thresholds only in ST, ST will not consider the scenario in which Bill retrieves large portions of both relations in one session, which would be anomalous.

To overcome this drawback, we introduce SE, an offline AD approach that complements ST by evaluating users’ sessions at their ends. In the following, we describe the internal representation of the sessions’ information and the methods of the training and detection phases of operation of the basic SE approach.

A. Session Representation

Training sessions and sessions evaluated during the detection phase of SE are represented in the form of session features records (SFRs), which encode the aggregate selectivities of the referenced portions of the T-DB relations.

An SFR that represents a session s is an array of length equal to the number of relations stored in the T-DB. Each entry in this array contains a floating-point value that lies within the selectivity range, i.e., [0, 1], and represents a specific T-DB relation. The value that corresponds to a relation that has not been referenced in any select query executed during s is set to zero, whereas the value that corresponds to a relation r referenced in one or more select queries (Q) executed during s is set to the aggregate selectivity of access to r, i.e., the fraction of r that has been retrieved by Q; this value is computed by first executing the query:

$$Q_r\text{: SELECT COUNT(*) FROM } (< Q >),$$

Where $< Q >$ is a list of queries whose length is equal to the number of queries in Q. If the number of queries in Q is equal to n, $< Q >$ has the form: $< Q_{r_1} >$ UNION $< Q_{r_2} >$ ... UNION $< Q_{r_n} >$ and its purpose is to combine the result-sets of the queries $\{< Q_{r_1} >, ..., < Q_{r_n} >\}$ and remove duplicate rows. $< Q_{r_i} >$ selects the primary-key values of the rows retrieved by the i-th query in Q and has the form:

$$Q_{r_i}\text{: SELECT } < pk > \text{ FROM } < R > \text{ WHERE } < p >,$$

Where $< pk >$ is replaced with the primary-key attribute(s) of r, $< R >$ is replaced with the set of relations referenced in the i-th query in Q, $< p >$ is replaced with the predicate of the i-th query in Q. The selectivity of the fraction of r accessed in s is then computed as the result of the division between the result of Q_r and the cardinality of r.

B. Algorithms

For constructing of the roles' profiles, SE populates the SFRs that represent the sessions owned by the users of each T-DBMS role, summarizes them into clusters, and maintains a mapping between each role and the clusters to which the sessions owned by the role's users belong. We refer to this mapping as the roles-clusters mapping

(M_{R-C}). The clusters' information and the roles-clusters mapping compose the final roles' profiles.

SE evaluates the activities performed during each session and considers a session anomalous in two cases: (1) if it is an outlier to the AD model, or (2) if it is not expected from a user of the role of the owner based on the values of the features captured in the session's SFR. Algorithm 5.3 shows the steps followed by the A-Detector for the inspection of a session s.

```
Algorithm 5.3: Detection Phase of SE
Input: queries executed during a session $s$.
Output: whether $s$ is anomalous or normal.
1. Compose the SFR ($sfr$) that characterizes the features of $s$.
2. Employ a novelty detection algorithm to find if $sfr$ is an outlier of the training
   clusters. If $sfr$ is found to be an outlier, flag $s$ as anomalous and terminate.
3. Employ the clustering algorithm used in training to find the training cluster
   ($c$) to which $sfr$ belongs.
4. Look up $M_{R-C}$ to find the set of clusters ($C$) to which the role of the session's
   owner corresponds.
5. If $c \in C$, consider the session normal. Otherwise, flag the session as anomalous.
```

5.3.2 Session Evaluation with Lengths' Partitioning

An important feature of the sessions that is not considered by the basic SE approach is the sessions' lengths. This feature is fundamental because it directly impacts the aggregate selectivites of the relations stored in the SFRs upon which SE relies for AD. During long sessions, the T-DBMS users are expected to issue more queries than during shorter sessions. The aggregate selectivities of queries executed during the long sessions are thus expected to be higher. We propose extending the basic SE approach to take into account the sessions' lengths in AD. We refer to the new approach as SE+.

The training algorithm of SE+ processes the logs' sessions in two stages. In the first stage, SE+ uses a uni-dimensional clustering algorithm to partition the range of the training sessions' lengths. Uni-dimensional clustering works by summarizing sessions that have comparable lengths into one cluster, which represents a sub-range of the full range of the input lengths. In the second stage, SE+ follows the basic SE approach for clustering sessions whose lengths belong to the same partition generated by the first stage. SE+ maintains a mapping between the generated clusters and the roles of the sessions' owners as discussed in the basic SE approach.

The training profiles are a list of quadruplets. Each quadruplet represents a partition generated by the first stage of the training algorithm of SE+ and has the form: $(r_{\text{start}}, r_{\text{end}}, C, M_{R-C})$, where $[r_{\text{start}}, r_{\text{end}}]$ is the length sub-range that the quadruplet represents, C is the clustering information of the SFRs whose lengths lie within $[r_{\text{start}}, r_{\text{end}}]$, and M_{R-C} is the mapping between the roles and the training clusters generated for the sub-range.

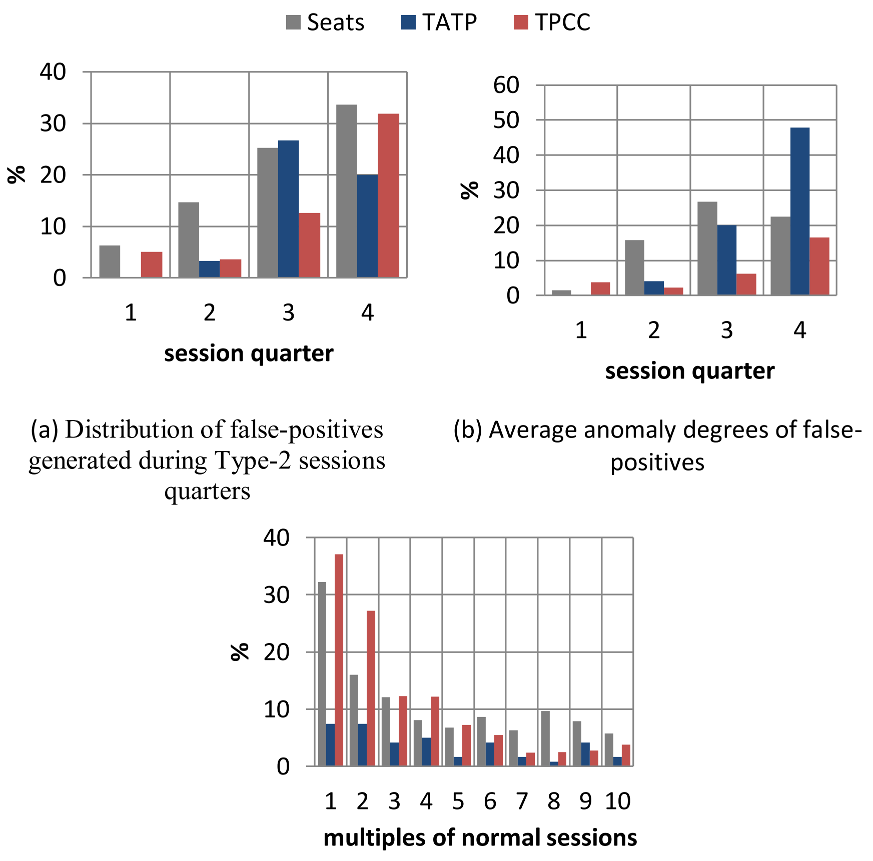
SE+ considers a session anomalous in three cases: (1) if its length is different from the lengths of the training sessions, (2) if the aggregate selectivities encoded in its SFR do not resemble those of sessions of comparable lengths, or (3) if it is not expected from a user of the role of the owner. The evaluation of a recently-ended session in SE+ is performed in two main stages. In the first stage, which is referred to as Stage 1 of SE+, training sessions whose lengths are comparable to s are located. The second stage, which is referred to as Stage 2 of SE+, is similar to the basic SE approach and compares the SFRs of the training sessions found in the first stage to the SFR of s.

5.4 Experimental Evaluation

We now present the results of experimental activities for the evaluation of our AD techniques. In the implementation, we used the optimizer that is part of the PostgreSQL DBMS and an SQL parser developed by us. The main metrics we considered in the evaluation are the false-positive and false-negative error rates.

Since there are no publicly available datasets associated with ground truth information that contain DB misuse scenarios [32], we relied upon the OLTP-Benchmark [31] workloads for generating the datasets used in the evaluation.

The OLTP-Benchmark contains different workloads; each of which operates on a pre-designed DB that is loaded with data generated by the benchmark's programs. A workload contains one or more DB access scenarios; each contains one or more queries. The number of users/roles, referred to as terminals in the benchmark's code-



(a) Distribution of false-positives generated during Type-2 sessions quarters

(b) Average anomaly degrees of false-positives

(c) Distribution of false-negatives

Figure 5.2.: Results of the evaluation of ST

base, the workload distribution across the different scenarios, and the rate of query submission are configuration parameters supplied to the benchmark at the start of its execution. Therefore, we could configure the benchmark's workloads for the purpose of generating data that applies for different testing and experimentation scenarios.

The rest of this section is divided into four subsections in which we present and discuss the results of the evaluation of ST, SE and SE+, and finally make some concluding remarks.

5.4.1 Results of the Evaluation of ST

In addition to reporting the false-positive and false-negative error rates in the result of the evaluation of ST, we also report the degrees of anomalies associated with the generated false-positives and the selectivities of portions of the T-DBMS relations that are returned to insiders as a result of the occurrence of false-negatives.

To generate the sessions used for training ST, we ran the benchmark's TPCC, Seats and TATP scenarios for 10 time units setting the number of terminals, i.e., user sessions, to 80. We gave the different scenarios equal weights and set the query submission rate to 10 queries/time unit. We refer to the training sessions in the rest of the discussion as Type-1 sessions.

A. False-Positives

In order to measure the false-positive errors, we generated 20 user sessions of lengths similar to that of Type-1 sessions; we refer to these sessions as Type-2 sessions. Since these sessions are similar to Type-1 sessions in both length and query submission rate, we assumed that all queries in Type-2 sessions are normal. Therefore, we counted the anomalies generated by the A-Detector upon the inspection of Type-2 sessions' queries as false-positives.

The result shows a low average false-positive error that ranges between 10% and 20% for the different workloads. However, this result is not enough to conclude that ST can be practically employed; it is also important to consider the time when these anomalies are generated because anomalies are likely to be inspected by a human security administrator and the generation of a large number of anomalies in a short time period is impractical.

Figures 5.2(a) and 5.2(b) show the distribution of the generated false-positives over the time course of Type-2 sessions and the average anomaly degrees of the anomalous queries for the different workloads. We assumed that the evaluation of query sequences is performed directly after the execution of each query. The result indicates higher errors at the end of the sessions; that is when the threshold levels are most likely reached. The average anomaly degrees of false-positives are rather low and only reach 25% above the selectivity thresholds.

B. False-Negatives

In order to measure the false-negative errors, we generated 20 user sessions similar to Type-1 and Type-2 sessions. We submitted the sessions' queries to the A-Detector for inspection after running Type-2 sessions maintaining the state of the AD data-structures. Since these queries are atypical because they follow the end of Type-2 sessions, we considered queries that are found to be normal by the A-Detector as false-negatives. We excluded queries that access the same data as any of Type-2 sessions' queries. We refer to the sessions used to measure the false-negative errors as Type-3 sessions.

The result shows a low average false-negative error that ranges between 4% and 12% for the different workloads. Type-3 sessions used in this set of experiments are ten times longer than Type-1 and Type-2 sessions. Since the average false-negative error depends on the sessions' lengths, it is important to measure the error over the time course of Type-3 sessions. Figure 5.2(c) shows the average false-negatives for Type-3 sessions' lengths that are multiples of the lengths of Type-1 and Type-2 sessions. The result shows that later queries in Type-3 sessions are more likely to be detected. Although high error occurs at small multiples, this result does not indicate a major security issue because the anomalous queries that are not detected retrieve small portions of the referenced relations whose selectivities are less than 2%.

5.4.2 Results of the Evaluation of SE

We selected five scenarios of the TPCC workload and four scenarios of the Seats workload to use as datasets for the evaluation of the basic SE approach. We considered each workload individually in the evaluation and assumed that one T-DBMS role executes each workload scenario. We used the implementation of the machine learning techniques of the Python library Scikit-learn [35].

A. False-Positives

To measure the false-positive errors, we ran each workload scenario for 10 time units and set the query submission rate and the number of terminals/sessions to 10 and 100, respectively. We used 80% of these sessions for training and the remaining 20% for evaluation. We refer to the training sessions and the evaluation sessions as Type-1 and Type-2 sessions, respectively. Since all Type-2 sessions have similar characteristics as Type-1 sessions used in training, we considered the anomalies generated by the A-Detector upon evaluating Type-2 sessions as false-positives.

We evaluated the performance of SE when Robust Covariance [36] and Isolation Forests [37] algorithms are employed in the implementation of Step 1 of SE and K-Means [38], Mean Shift [39], and Affinity Propagation [40] are employed in the implementation of Step 3. We set the parameters of K-Means and Affinity Propagation to 8 and 0.5, respectively, and the contamination parameter of both novelty detection algorithms to 0.0001 to indicate that the training data contains no outliers.

The result of the evaluation shows that when using different dataset sizes that range between 15% and 100% of the number of generated Type-1 sessions for training, less than 5% false-positives are produced by SE. These errors occur as a result of Steps 3-5 of the detection phase algorithm where the evaluation instances are not properly attributed to the correct roles.

B. False-Negatives

We use the following distance metrics to describe the distance between normal sessions used for training and anomalous sessions used for measuring the false-negative errors produced by SE.

(a) *Minimum Total Selectivity Distance.* The minimum total selectivity distance between an anomalous testing session s_{test} and the training sessions owned by a T-DBMS role r is denoted as $D_s(s_{\text{test}}, r)$ and is computed by comparing the values in the SFR of s_{test} to the corresponding values in the SFRs of the training sessions of r. Formally,

$$D_s(s_{\text{test}}, r) = \min_{\forall s_{\text{train}} \in S_r} d_s(s_{\text{test}}, s_{\text{train}}),$$

Where S_r is the set of training sessions of r, s_{train} represents one of these sessions and $d_s(s_{\text{test}}, s_{\text{train}})$ is the total selectivity distance between the two sessions s_{test} and s_{train} and is computed as:

$$d_s(s_{\text{test}}, s_{\text{train}}) = \sum_{i \in |\text{SFR}(s_{\text{test}})|} abs(\text{SFR}_i(s_{\text{test}}) - \text{SFR}_i(s_{\text{train}})),$$

Where $\text{SFR}_i(s_k)$ is the i-th selectivity value in the list of aggregate selectivities stored in the SFR of s_k.

(b) *Minimum Total Row Distance.* The minimum total row distance between an anomalous testing session s_{test} and the training sessions owned by a T-DBMS role r is denoted as $D_r(s_{\text{test}}, r)$ and is computed by comparing the number of rows retrieved from each DB relation by the queries executed during s_{test} and those executed during the training sessions of r. Formally,

$$D_r(s_{\text{test}}, r) = \min_{\forall s_{\text{train}} \in S_r} d_r(s_{\text{test}}, s_{\text{train}}),$$

Where $d_r(s_{\text{test}}, s_{\text{train}})$ is the total row distance between s_{test} and s_{train} and is computed as:

$$d_r(s_{\text{test}}, s_{\text{train}}) = \sum_{i \in |\text{SFR}(s_{\text{test}})|} |t_i| * abs(\text{SFR}_i(s_{\text{test}}) - \text{SFR}_i(s_{\text{train}})),$$

Where t_i is the T-DB relation that the i-th items in $\text{SFR}_i(s_{\text{test}})$ and $\text{SFR}_i(s_{\text{train}})$ correspond to and $|t_i|$ is the cardinality of t_i.

To measure the false-negative errors of SE, we considered the following two anomaly scenarios.

(a) *Anomalous outliers.* Testing sessions where datasets larger than the data accessed in training sessions are retrieved are referred to as anomalous outliers.

To generate anomalous outliers, we composed sessions that have relatively the same length as Type-1 sessions, but have higher query submission rates than Type-1 sessions. We set the rate of query submission to 1.5, 2, 3, ..., 9 multiples of the query submission rate of Type-1 sessions; this resulted in higher aggregate selectivities of the evaluation sessions that are not necessarily directly proportional to the query submission rates.

Figures 5.3(a) and 5.3(b) show the distribution of the false-negatives generated by the A-Detector. The result shows that SE can better detect anomalous instances that are more distant from the training instances. This result also indicates that the distance metrics described above are adequate in representing the differences between the training and detection instances. In general, SE can detect outliers at a distance more than 7% selectivity and 10 rows from the training records.

Figure 5.3(c) shows the effect of tuning the contamination parameter of the novelty detection algorithm on the accuracy of SE in the detection of anomalous outliers.

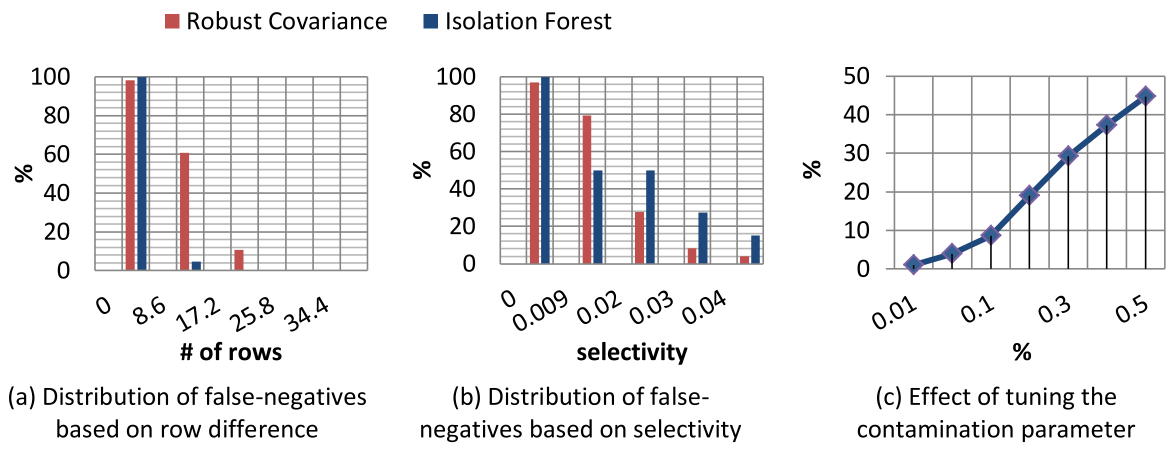


Figure 5.3.: Accuracy of the detection of anomalous outliers by SE

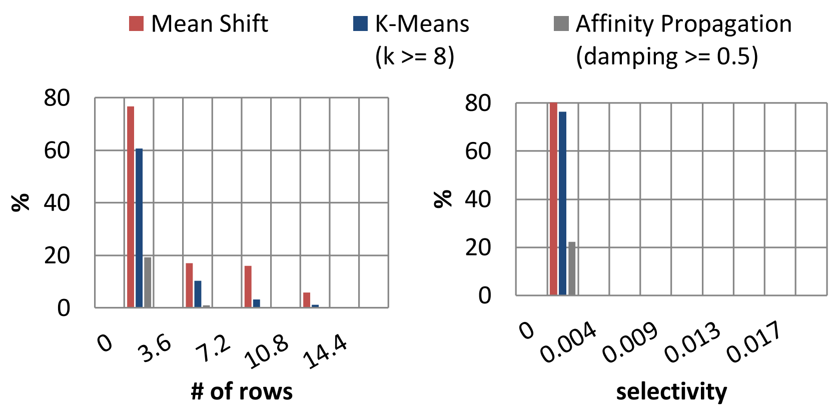
The number of false-negatives increases as the value of the contamination parameter increases due to the removal of some normal instances from the training data that support the AD model.

(b) *Anomalous inliers.* Anomalous inliers are sessions that are similar to the training sessions of one or more T-DBMS roles, but are owned by different roles. In other words, these sessions are considered inliers, i.e., normal when compared against the clustering model, but if the training data of the roles of their owners are considered, they should be considered anomalous.

To generate anomalous inliers, we used the training records for evaluation after changing the roles of the owners of the sessions that the records represent. Figure 5.4 shows the detection accuracy of SE when different clustering algorithms are employed in the implementation. The result indicates that SE could capture selectivity differences between the roles that are higher than 0.4%.

5.4.3 Results of the Evaluation of SE+

We employed Jenk's natural breaks clustering algorithm [41] and Robust Covariance novelty detection algorithm in the implementation of Stage 1 and Stage 2 of SE+, respectively. The inputs to Jenk's clustering are the lengths of the training sessions and the number of bins/breaks that the total range of the sessions' length are split into. The algorithm's output is sub-ranges of the input range of sessions'



(a) Distribution of false-negatives based on row difference

(b) Distribution of false-negatives based on selectivity difference

Figure 5.4.: Accuracy of the detection of anomalous inliers by SE

lengths. The value of the input number of breaks has to be chosen carefully as it directly impacts the accuracy of AD. We used trial-and-error to select the number of breaks for Jenk's clustering that properly fits the input training sessions by sequentially trying different values for the number of breaks and stopping the trials when the goodness of variance fit (GVF) [41] of the resulting breaks exceeds 90%.

Since Stage 2 of SE+ is similar to SE, we only considered measuring the false-positives produced by Stage 1. Sessions whose lengths are comparable to the lengths of the training sessions that are considered anomalous by Stage 1 of SE+ were counted as false-positives. To measure this type of error, we generated training and evaluation sessions by running the TPCC and Seats workloads for a time interval of length equal to 10 units. For each workload scenario, we generated 100 terminals/sessions; 80% of which were used for training and the remaining 20% were used for testing. We observed that none of the testing sessions was considered a false-positive. However, when evaluating the model of SE+ against the training sessions, less than 1% of these sessions were considered anomalous, i.e., false positives.

5.4.4 Concluding Remarks

- Since ST uses hard threshold values to differentiate between normal and anomalous behavior, considering the risk degrees of anomalous queries is useful for ruling out potential false-positives.

- SE and SE+ can accurately detect both unexpected data misuse scenarios and also capture the difference between the behaviors of the users of the different roles. They can thus prevent some data masquerading attacks and overcome errors in role definition and assignment.
- The distance metrics that describe the distance between normal and anomalous sessions are indicative of the risk degrees of anomalous sessions and can be used to decide on the response to the sessions' owners.
- Tuning the configuration parameters of the algorithms is essential for the correct operation of the AD techniques; this can be achieved using cross-validation where the training data is successively split into training and validation sets and trial-and-error is used to select proper values of the configuration parameters that produce the least error rates.

5.5 Conclusions

In this chapter, we presented techniques for the detection of knowledge aggregation attempts by insiders. Our techniques are designed to be practically adopted in DB systems and, based on our experimental evaluation, they can accurately detect anomalous behavior.

6 DETECTION OF ANOMALIES IN RATES OF TABLES REFERENCES AND TUPLES RETRIEVALS

In this chapter, we present AD techniques for DB access monitoring that aim at the detection of the following misuse scenarios.

1. *Data aggregation.* AD fails to detect data aggregation attempts if each query is inspected individually. As described in Chapter 5, tracking the aggregate sizes of result-sets of queries will rather be effective in this case. However, if the insider does not have prior knowledge on the distribution of the target data, many of his/her queries may result in retrieving no data or small amounts of data; therefore, the approach that relies on tracking aggregate sizes of result-sets of queries will also have limited detection accuracy or long time to AD. A better approach would be to track the users' rates of referencing the DB tables.
2. *Attempts to track data updates.* A malicious insider may execute one or more queries repeatedly across a temporal interval aiming at tracking updates to the data tuples read by the queries. These queries are considered legitimate if the insider has permissions to read from the DB entities referenced by the queries. However, the insider's behavior is anomalous when the access rates to the tuples retrieved by the queries are compared to the normal access rates by the insider or the users who belong to his/her role.

Our techniques rely on tracking the data access rates by the users of the T-DB for the prupose of the detection of the anomaly scenarios described above. The normal data access rates are captured from past logs of user activity during a training phase. This information is used to build profiles that describe the data access patterns of the DB users. After the training is complete, queries executed against the monitored DB are inspected in order to track the users' rates of referencing the DB tables and tuples. An increase in a user's data access rates beyond the normal levels is flagged as anomalous to indicate that the behavior of the user is suspicious and requires further analysis.

Our techniques inspect each user query in two main steps. The first step is referred to as preliminary inspection and aims at detecting anomalies in the rates of referencing the DB tables. Preliminary inspection of a query is performed before the query execution and is designed to be fast by only requiring parsing queries in order to extract their syntactic features.

The second step for query inspection aims at detecting anomalies in the rates of tuples retrievals and is referred to as deep inspection. Deep inspection of a query checks the raw data tuples retrieved by the query and thus requires the execution of the query against the monitored DB. Since the execution of a query for the purpose of AD only is not a suitable approach, we introduce an architecture that supports the inspection of the rows in queries result-sets before returning them to the issuers.

We implemented the proposed techniques and developed a proof of concept prototype of the architecture that shows that our techniques can be employed as part of DBMSs that return query result-sets in the form of pipe-lines of rows. We evaluated the proposed techniques using the query logs of a real DB. We present and discuss the results of the evaluation, which indicate that our techniques can accurately detect anomalies in data access rates and produce few false alarms. Based on the results of the evaluation, we draw conclusions on approaches for choosing the system configuration parameters and for estimating the risk degrees of queries and user activities [5].

The remainder of this paper is organized as follows. We discuss the architecture in which the proposed techniques can be deployed in Section 6.1. In Sections 6.2 and 6.3, we present detailed algorithms for the proposed techniques and the data structures that support their implementation. We describe our approach for the experimental evaluation of the proposed techniques and discuss the results of the evaluation in Section 6.4. Section 6.5 concludes the chapter and discusses potential future work.

6.1 Architecture

The proposed techniques operate in two phases: training and detection. During the training phase, a Profiler module is fed with past logs of the T-DBMS in order to capture the T-DB users' access patterns and build profiles of the users accordingly. The Profiler executes the training queries on a copy of the T-DB, referred to as the training DB. The training DB also stores temporal data that characterize references

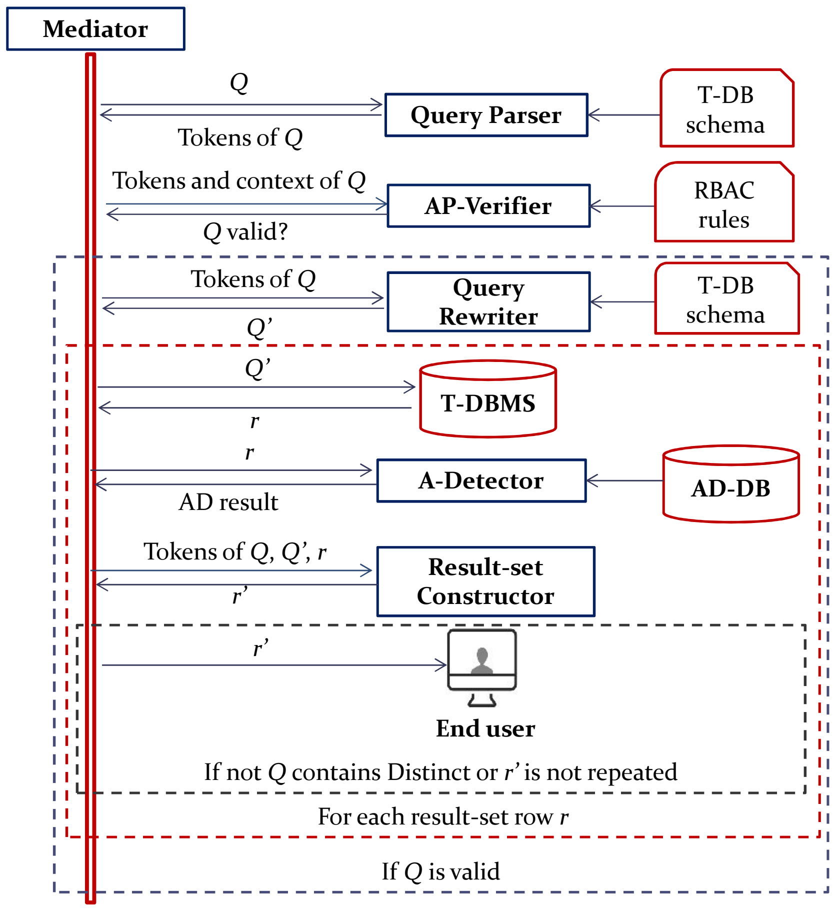


Figure 6.1.: Steps for the inspection of the result-set rows of a new user query (Q)

to the T-DB tables and tuples. A copy of the T-DB is thus required at the start of training and is used to setup the training DB. The Profiler combines individual user profiles to form roles profiles, which are smaller in number and can thus be better managed.

The training is done once offline and transparently to the T-DBMS users. The detection phase starts after the training is complete. The communication between the T-DBMS and its users during the detection phase is established through the use of a Mediator component. The Mediator executes queries on behalf of the T-DB users and is thus responsible for logging users interactions with the T-DBMS in addition to the AD results.

The Mediator communicates with the different system components to detect anomalies in the users behaviors based on the select queries executed by the users. The Mediator employs an SQL parser to extract the syntactic features of each select query, which characterize the tables and attributes referenced in the query, in order to prepare it for inspection. The Mediator thus has access to a copy of the schema of the T-DB. It is the job of the Mediator to monitor other types of queries that lead to changes to the T-DB schema and data.

Given an input select query, the Mediator first verifies that the issuer has the necessary privileges to access the DB entities referenced in the query. For this purpose, the Mediator employs an Access Privileges Verifier component (AP-Verifier), which refers to the RBAC rules of the T-DBMS to decide whether the query is allowed by its issuer. The Mediator generates an error and stops the execution of the query if it does not match the RBAC rules. Otherwise, the Mediator sends the query to a Query Rewriter component, which performs modifications to the query string that are necessary for the extraction of additional information on the data retrieved by the query.

The mediator then executes the modified query at the T-DBMS on behalf of the query issuer and establishes a connection with an A-Detector component, which performs the actual inspection of the query. The Mediator sends the query to the A-Detector and relays the rows of the result-set of the query from the T-DBMS to the A-Detector. The A-Detector stores user tracking data and parts of the T-DB data in a DB referred to as the AD-DB and uses this DB for query inspection.

Based on the result of the inspection and the rules of the AD response policies, the Mediator chooses whether to relay the rows of the result-set to the issuer. The Mediator employs a Result-set Constructor component to modify each result-set row to match the original input query and present the row in the format accepted by the issuing user application. The Mediator also removes any duplicate rows that are added due to the modifications made to the query before its execution.

The execution of queries on commercial DBMSs follows a pipelined approach in which the rows of the result-sets are presented as a pipeline rather than as a bulk. This approach is useful for the proper application of our techniques as the inspection of the tuples in the output of a query at the A-Detector can be performed in parallel to the execution of the query at the T-DBMS.

Figure 6.1 shows the interactions between the system components for the purpose of the inspection of a new user query.

6.2 Training Phase

The purpose of training is to measure the expected number of references by the users of each role to the T-DB tables and tuples during tracking intervals of different lengths and store this information in roles profiles. The profile of a role consists of two sets of records. The first set represents the expected rates for referencing the T-DB tables in queries, while the second set represents the expected rates of tuples retrievals. Table 6.1 shows an example record set. The attribute t refers to the name of a T-DB table, l_{i} refers to the length of tracking intervals and n refers to the number of expected references to t or to the tuples of t within a time interval of length equal to l_i. The lengths of time intervals are measured in terms of the system resolution (L_{res}).

Our approach for computing the rates of retrievals of the T-DB tuples is to not distinguish between retrievals of a tuple by the same user if the time-stamps of the retrievals are within an interval of length equal to L_{res}. We made this choice based on our experiments on a real DB referenced by an application program. We noticed that the program may read the same tuple multiple times by different queries as part of the program's flow. A tuple may also be read several times by the same query as a result of a primary-key-foreign-key relationship between tables joined by the query. This resolution assumption complies with our initial goal to detect attempts for tracking data updates as tuples are unlikely to change within short time intervals. Reading the same tuple multiple times within a short time interval thus does not convey additional information to a malicious insider.

On the other hand, the resolution concept does not apply to tables references as multiple references to the same table within a short time interval that return no data or small amounts of data can convey important information on the distribution of the data stored in the table. The main design goal of our techniques is to detect this case.

During the training phase, the Profiler processes the T-DBMS logs in two main steps: organizing logs and building profiles in order to form roles profiles. In the rest of this section, we discuss details on the two training steps.

Table 6.1.: Data access rates

t	l_{i}	n
Emps	2	1
Emps	3	2
Emps	4	2

Table 6.2.: Example DB table

eid	name	position	salary
1	Lucas Isaac	JD1	50,000
2	John Blake	JD2	62,000
3	Jamie Adam	JD2	65,000
4	Joseph King	SD1	80,000

Table 6.3.: Example roles and users sub-logs

(a) Role sub-log

Query	uid	Timestamp

SELECT * FROM Emps
WHERE name = ‘Lucas Isaac’
	1001	6/1/17 9:15:35:010

UPDATE Emps
SET salary = salary + 500
WHERE name = ‘Lucas Isaac’
	1002	6/1/17 9:30:48:123

SELECT * FROM Emps
WHERE name = ‘John Blake’
	1001	6/1/17 9:50:52:820

SELECT * FROM Emps
WHERE name = ‘John Blake’
	1002	6/1/17 9:55:00:010

SELECT * FROM Emps
WHERE name = ‘John Blake’
	1002	6/1/17 9:58:00:020

(b) A user sub-log (uid = 1001)

Query	Timestamp

SELECT * FROM Emps
WHERE name = ‘Lucas Isaac’
	6/1/17 9:15:35:010

UPDATE Emps SET salary = salary + 500
WHERE name = ‘Lucas Isaac’
	6/1/17 9:30:48:123

SELECT * FROM Emps
WHERE name = ‘John Blake’
	6/1/17 9:50:52:820

Table 6.4.: Maintenance of time-series by result-based AD
($L_{\text{res}} = 20$ mins, $L_{\text{i}} = \{2, 3\}, L_{\text{s}} = 3$)

Query Time	ats	sums	Sample Sums
9:01	8:02 8:22 8:42 9:02 0 0 1	{1,1}	{}
9:50	9:02 9:22 9:42 10:02 1 0 1	{1,2}	{(l_{i} = 2) > {1,1}, (l_{i} = 3) => {1,1,1}}
10:01	9:02 9:22 9:42 10:02 1 0 1	{1,2}	{(l_{i} = 2) => {1,1}, (l_{i} = 3) => {1,1,1}}
10:03	9:22 9:42 10:02 10:12 0 1 1	{2,2}	{(l_{i} = 2) => {1,1,1}, (l_{i} = 3) => {1,1,1,2}}

6.2.1 Organizing Logs

The Profiler organizes the training logs by partitioning their queries into roles sub-logs and users sub-logs. Users sub-logs contain sufficient data that allows the Profiler to process them individually and sequentially in the profiles building step.

Definition 6.2.1 *The sub-log of a role r contains all select queries issued by the users activating the role r at the time of executing the queries in addition to all other types of queries present in the logs. Queries in a role sub-log are ordered chronologically.*

Definition 6.2.2 *The sub-log of a user u who belongs to a role r contains all select queries in r's sub-log that are executed by u in addition to all other types of queries present in the logs. Queries in a user sub-log are ordered chronologically.*

Tables 6.3a and 6.3b show an example role sub-log and one of the corresponding users sub-logs respectively. The logs contain queries that reference the table *Emps* (short for employees), which has the schema: $Emps(\underline{eid}, name, position, salary)$.

6.2.2 Building Profiles

The Profiler computes the data access rates for each role based on the sub-logs of its users. For this purpose, the Profiler references the training DB to execute queries and record access time information. The training DB is initialized before processing each user sub-log to restore the initial state of the T-DB at the start time of the logs.

The training DB contains intermediate tracking information. The access rates to tables are stored in a training DB table named: *tables-references*. A record in *tables-references* contains four attributes:

1. *table-name*: the name of a T-DB table,

2. *ats* (short for *access-time-series*): a time-series that stores the references made by the user whose sub-log is currently being processed to the table with the name *table-name*,

3. *ats-start*: the time-stamp of the start of *ats*, and

4. *sums*: a list that contains the sums of entries in *ats* that lie within tracking intervals of different lengths.

Other tables in the training DB correspond to the T-DB tables. A table t_{train} that corresponds to the T-DB table t contains all the attributes of t and the attributes *ats*, *ats-start* and *sums* that are similar to the temporal attributes of *tables-references*.

Given a user sub-log, the Profiler scans its queries and directly executes all statements on the training DB except select statements. When a select query is encountered, the Profiler composes and executes a modified version of the query against the training DB by replacing the projection-list of the original query with the list of primary-keys of the tables referenced in the query. This information is used to identify the raw tuples retrieved by the query. For example, if the original query is:

```
SELECT name
FROM Emps
WHERE salary ≥ 60,000,
```

the modified version of the query will be:

```
SELECT eid
FROM Emps
WHERE salary ≥ 60,000.
```

The Profiler then scans the result-set values in the primary-key attributes of each table and updates the time-series that correspond to each value. The Profiler also records the time-stamp of the query in *tables-references*.

To maintain the training DB time-series and record references to the T-DB tables and tuples, the Profiler employs a sliding-window algorithm. When the time-stamp of a training query does not fall within the interval initially-represented by a time-series stored in a training DB tuple, the time interval that the time-series represents should be updated, i.e., the time-series entries should be shifted and older entries should be discarded. To avoid recomputing the number of references that lie within the shifted interval, the sliding-window algorithm is used to maintain the sum of references encoded in the part of the interval that is not discarded. Samples of the sums of references encoded in discarded time-series in *tables-references* and other training tables are recorded in the training DB. Table 6.4 shows the steps for maintaining a time-series based on the time-stamps of queries that reference the table or tuple that the time-series represents.

Recording references to tuples by aggregate select queries is performed differently based on the type of the aggregate functions of the queries as follows.

1. *TOP n queries.* A TOP n query selects the first n rows only from the result-set of a similar query that does not contain the TOP n function. Given a TOP n training query, the Profiler modifies the query by removing the aggregate function from the query and adding the primary key attributes of the tables referenced by the query to the query's projection list as previously explained for non-aggregate queries. The Profiler then executes the query, observes the output rows in its result-set, records the original query's time-stamp in the time-series of the corresponding T-DB tuples and stops the execution of the query after n rows are observed.

2. *MAX and MIN queries.* The Profiler executes the original MIN and MAX queries and then executes additional queries to extract the primary-keys of the tuples referenced by the original queries. For example, given the query:

   ```
   SELECT MAX(salary)
   FROM Emps,
   ```

   the profiler executes the query as is and observes the output value of the MAX function; in this query case, the result is equal to 80,000 based on the T-DB data shown in Table 6.2. The Profiler then records the time-stamp of the original training query in the time-series of tuples that correspond to the rows of the result-set of the query:

```
SELECT eid
FROM Emps
WHERE salary = 80,000.
```

3. *Other aggregate functions, e.g., COUNT, SUM, AVG, and STDEV.* The Profiler discards all types of aggregate queries other than TOP n, MAX and MIN queries as they present data to the issuer that may have been computed based on a large number of values of the T-DB attributes; marking the tuples that contain these values as read will be inadequate as the attributes of these rows may not be contributing much to the results of the original queries.

After processing the sub-logs of all the users of one role (rl), the Profiler computes the threshold levels to be stored in rl's profile. Since the training data may contain anomalies, the maximum values of the sample sums of time-series are inadequate measures of the threshold levels. Alternatively, given a list of values that represents the sample sums of the time-series of references to a table t or the tuples of a table t by the users of rl during time intervals of length l, the Profiler considers the p_{upper} percentile of the values as the threshold level for referencing t or the tuples of t and stores this information in record-sets similar to the ones shown in Table 6.1. In addition to removing previous anomalous references to tables and tuples from the training data, using the p_{upper} percentiles as thresholds is also beneficial in eliminating the effect of variations in training references rates on the accuracy of computation of the training thresholds.

Algorithm 1 shows parts of the training phase algorithms. Algorithm 1 references the configuration parameters listed in Table 6.5. Procedure *record-tuples-retrievals* describes the steps followed by the Profiler for processing a list of primary-key values of a table that are part of the result-set of a training select query. For each primary-key value in the input list, the procedure extracts the training DB tuple (r) that corresponds to the primary-key value (line 3), finds the bucket that corresponds to the query execution time in the time-series stored in r (line 4) and updates this time-series to include the query execution time if necessary (lines 5:16). Time-series related to the T-DB tables and stored in *tables-references* are updated in an approach

Table 6.5.: Configuration parameters of result-based AD

Name	Description	Default
L_{res}	Resolution	1 hour
N_{i}	Number of tracking intervals lengths	1
L_{i}	An array of length equal to N_{i} that stores the lengths of the tracking intervals	$\{6\}$
L_{s}	Length of time-series in the training and AD DBs	10
p_{upper}	Upper percentile (used for choosing training thresholds)	95%

Algorithm 1 Training Phase Algorithms

```
1: procedure RECORD-TUPLES-RETRIEVALS($t$: name of a T-DB table, $PK$: list of values of the primary-key of $t$, $query\text{-}time$: the time of execution of a query that retrieves the tuples identified by the values in $PK$)
2:     for each $pk \in PK$ do
3:         $r \leftarrow$ read-tuple$(t, pk)$
4:         $index \leftarrow \frac{query\text{-}time \; - \; r.ats\text{-}start}{L_{\text{res}}}$
5:         while $index >= L_{\text{s}}$ do
6:             $index$- -
7:             for $i = 0 : N_{\text{i}} - 1$ do
8:                 $first \leftarrow r.ats[L_s - L_{\text{i}}[i]]$
9:                 record-sum$(t, L_{\text{i}}[i], r.sums[i])$
10:                $r.sums[i] \leftarrow r.sums[i] - first$
11:            $r.ats\text{-}start \leftarrow r.ats\text{-}start + L_{\text{res}}$
12:        if $r.ats[index] == 0$ then
13:            $r.ats[index]$++
14:            for $i = 0 : N_{\text{i}} - 1$ do
15:                if $index \geq L_s - L_{\text{i}}[i]$ then
16:                    $r.sums[i]$++
```

similar to *record-tuples-retrievals* except that the concept of resolution employed in *record-tuples-retrievals*:12 does not apply in case of tables.

6.3 Detection Phase

During the detection phase, the A-Detector tracks the access rates of the users to the T-DB tables and tuples and flags an anomaly if any of the thresholds captured during the training phase is exceeded. The A-Detector performs the inspection of each input select query in two main steps: preliminary inspection and deep inspection. The goal of preliminary query inspection is to check if the thresholds on tables references rates have been exceeded, while the goal of deep inspection is to check if the thresholds on tuples retrievals rates have been exceeded.

The A-Detector uses the AD-DB to store information on references to the tables and tuples by the T-DB users for the purpose of query inspection. References to the T-DB tables are stored in a table named *tables-references*. Each table t in the T-DB has two corresponding tables in the AD-DB: t*-PK* and t*-tuples-retrievals*, which store the primary-key values of t and the time-stamps of tuples retrievals respectively. The time-stamps of user queries are stored in an approach similar to the organization of time-stamps in the training DB.

The schema of the AD-DB is shown in Figure 6.2. Tables 6.7a and 6.7b show the AD-DB tables that correspond to the T-DB table *Emps* shown in Table 6.2. The time-series in *Emps-tuples-retrievals* correspond to the log queries in Table 6.3a.

Preliminary inspection of an input query requires parsing the query only in order to extract its range-tables, i.e., the tables referenced in the query. The A-Detector records the time-stamp of the query in the time-series in the AD-DB table *tables-references* and returns the result of the preliminary inspection to the Mediator. Since the tuples retrieved by the input query cannot be located based on the query's result-set individually, deep inspection of an input query requires the execution of a modified version of the query against the T-DB. The Mediator composes a new query by adding the primary-keys of the query's range-tables to the projection-list of the query to extract raw tuple information from the T-DB. For example, if the original query is:

SELECT name
FROM Emps
WHERE salary $\geq$ 60,000,

the modified version of the query will be:

SELECT eid, name
FROM Emps
WHERE salary $\geq$ 60,000.

The Mediator then executes the modified query against the T-DB, observes the output rows in the result-set and relays them to the A-Detector.

Given one result-set row of a query under inspection, the A-Detector finds the values of the primary-keys of each range-table, checks if any of the thresholds on the retrievals of tuples from the table has been exceeded by the issuer, and records the time-stamp of the new retrieval in the AD-DB. The A-Detector responds to the Mediator with the result of AD, which in-turn updates the tuple to match the original input query.

It is to be noted that the approaches for extracting the raw tuples retrieved by training queries and queries under inspection are different. The original projection-list of a query under inspection is retained because the values of the projection-list attributes are relayed to the query issuer after performing AD; these values are not useful in the case of a training query. However, the modifications made to queries under inspection not only change the result-sets rows, but may also result in adding more rows to the result-sets. Tables 6.6a and 6.6b illustrate the differences between the result-set of an example query that has the syntax:

SELECT DISTINCT position
FROM Emps
WHERE salary $\geq$ 60,000

and the result-set of its modified version that has the syntax:

SELECT DISTINCT eid, position
FROM Emps
WHERE salary $\geq$ 60,000.

It is the job of the Mediator to detect and discard added rows. For this purpose, when given a select distinct query, the Mediator uses a hash-table session variable to record the combined values of the attributes of rows sent to the user as part of the result of the query, searches for each row in the query's result-set and drops rows that are found in the hash-table.

Algorithm 2 shows parts of the detection phase algorithms. Procedure *check-tuples-retrievals* in Algorithm 2 shows the steps for the inspection of the primary-key

Table 6.6.: Result-sets of an example query and its modification after adding the primary-keys of the range-tables

(a) Result-set of the original query

position
JD2
SD1

(b) Result-set of the modified version

eid	position
2	JD2
3	JD2
4	SD1

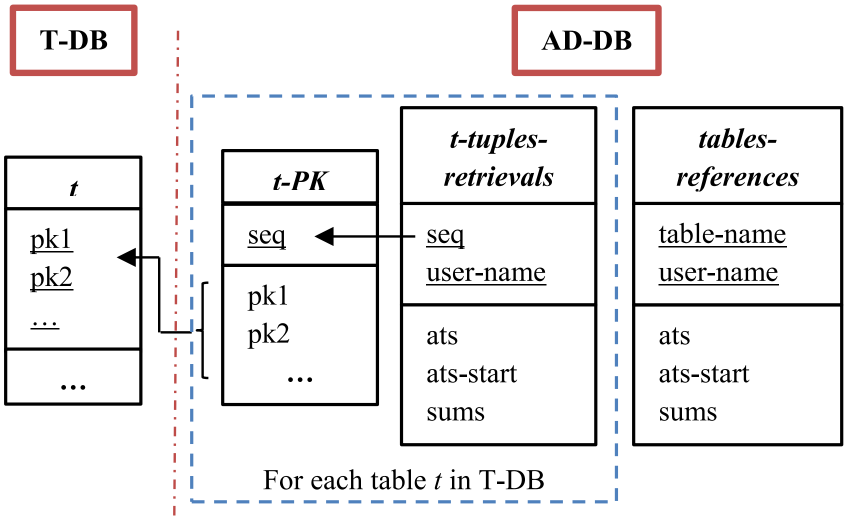


Figure 6.2.: Data-structures used during the detection phase of result-based AD

values of a range-table of an input query. The algorithm flags an anomaly if the retrievals of one or more tuples are found to be anomalous. Other approaches for estimating risk degrees for queries and user activities are discussed in Section 6.5. The algorithm for checking the rates of referencing a range-table of an input query is similar to Procedure *check-tuples-retrievals*; the main difference between the two algorithms is that the use of resolution in *check-tuples-retrievals*:15 has to be omitted in case of checking tables references.

Table 6.7.: Detection phase data-structures corresponding to one DB table

(a) *Emps-PK*

seq	eid
0	1
1	2
2	3
3	4

(b) *Emps-tuples-retrievals*

seq	user-name	ats	ats-start	sums
0	1001	{0,0,1}	6/1/17 8:16:00:000	{1,1}
1	1001	{0,0,1}	6/1/17 8:51:00:000	{1,1}
1	1002	{0,1,1}	6/1/17 9:16:00:000	{2,2}

Algorithm 2 Detection Phase Algorithms

```
1: procedure CHECK-TUPLES-RETRIEVALS($query\text{-}time$: the time of execution of the input query $q$, $u$: the name identifier of the issuer of $q$, $rl$: the name identifier of the role activated by $u$ at the time of executing $q$, $r$: a row in the result-set of $q$, $T$: range-tables of $q$)
2:     $result \leftarrow$ NORMAL
3:     for each $t \in T$ do
4:         $thr_t \leftarrow$ get-tuples-retrievals-threshold$(rl, t)$
5:         $PK_t \leftarrow$ get-table-PK$(r, t)$
6:         $r_{\mathrm{AD}} \leftarrow$ get-AD-tuple$(u, t, PK_t)$
7:         $index \leftarrow \frac{query\text{-}time\ -\ r_{\mathrm{AD}}.ats\text{-}start}{L_{\mathrm{res}}}$
8:         while $index >= L_{\mathrm{s}}$ do
9:             for $i = 0 : N_{\mathrm{i}} - 1$ do
10:                $first \leftarrow r_{\mathrm{AD}}.ats[L_s - L_{\mathrm{i}}[i]]$
11:                $sums[i] = sums[i] - first$
12:            remove-first$(r_{\mathrm{AD}}.ats)$
13:            $index$- -
14:            $r_{\mathrm{AD}}.ats\text{-}start \leftarrow r_{\mathrm{AD}}.ats\text{-}start + L_{\mathrm{res}}$
15:        if $r_{\mathrm{AD}}.ats[index] == 0$ then
16:            $r_{\mathrm{AD}}.ats[index]$++
17:            for $i = 0 : N_{\mathrm{i}} - 1$ do
18:                if $index >= L_s - L_{\mathrm{i}}[i]$ then
19:                    $r_{\mathrm{AD}}.sums[i]$++
20:                    if $r_{\mathrm{AD}}.sums[i] > thr_t[i]$ then
21:                        $result \leftarrow$ ANOMALOUS
22:    return $result$
```

6.4 Experimental Evaluation

We now present the results of experiments for the evaluation of the proposed techniques. The data-set used in experimentation [8] is a real SQL Server DB that contains 71 data tables and the logs of queries executed by the users of an application program that references the DB. The query logs cover a time period of length equal to 3 days and contain 16 user sessions and about 6,000 select queries; each conforms with one of 220 query templates. The main metrics considered in the evaluation are the false-positive error rates (FPRs) and false-negative error rates (FNRs).

6.4.1 False-Positives

We assumed that the query logs contain no anomalies and thus considered the anomalies flagged by the A-Detector when inspecting any of the log queries as false-positive errors. We set the default value of p_{upper} to 95%; therefore, only 5% of the training sample sums related to a table or to the tuples of a table are discarded when computing the reference threshold related to the table or to its tuples.

We measured the effect of the following factors on the rate of the false-positive errors produced by preliminary and deep inspection.

1. *Size of the training data.*
   To generate training data-sets of different sizes, we chose p_t% of the total available training data for building profiles and the most recent 20% of the log queries for model evaluation. We measured the FPRs for p_t equal to 10, 20 ... 80.
   Figures 6.3(a) and 6.4(a) show the percentages of queries flagged as anomalous by preliminary inspection and deep inspection respectively for the different values of p_t. The error rates are computed as the average error rates produced as a result of using different tracking intervals lengths, which range between 2 and 8 hours. Preliminary inspection considers a query anomalous if one or more tables of the query's range-tables are flagged as anomalous as a result of the inspection of the query. Deep inspection considers a query anomalous if one or more tuples retrieved by the query are flagged as anomalous. Figure 6.4(b) compares the percentage of the number of tuples flagged as anomalous by deep inspection to the number of queries flagged as anomalous by the same method if one or more of the references to the tuples read by the queries were found to be anomalous.

The result proves that using more data for training produces a more accurate AD model and thus reduces the false-positive errors. Using 30% of the training data was sufficient for capturing accurate tables references thresholds and producing an accurate model for preliminary inspection. However, deep inspection required more data to produce an accurate model as deep inspection captures more information on data access rates.

To better understand the changes that occur to the characteristics of the training data when its size changes, we computed two statistical metrics for each of the training data-sets. These are the average number of references to the T-DB tables per hour and the average number of tuples retrieved per hour. Significant changes in the values of both metrics occur when the size of the training data-set changes. The values of the first metric stabilizes for training data-sets of sizes equal to or bigger than 30% of the total evaluation data-set as shown in Figures 6.3(b) and 6.3(c). The result thus explains why smaller training data-sets lead to more false-positive errors by preliminary inspection. However, the average number of tuples retrieved per hour was not suitable for describing the characteristics of the training data and none of the two metrics could be used to find the size of data that is suitable for training the deep inspection model.

In the rest of our experiments, we chose the default size of the training and evaluation queries to be 80% and 20% of the available queries, respectively. We were thus able to rule out the effect of insufficient training data on the accuracy of the AD model.

2. *Length of the tracking intervals.*

   Figures 6.3(d) and 6.4(c) show the average FPRs produced by preliminary inspection and deep inspection respectively for various lengths of the tracking intervals, which range between 2 and 8 hours.

   We observed that both preliminary inspection and deep inspection produce low error rates when the length of the tracking intervals is equal to 4 hours or more. These values are related to the lengths of the training sessions, which range between 2.5 and 5.5 hours. We can thus conclude that as the length of the tracking intervals is closer to the lengths of user sessions, more accurate thresholds on the rates of tables references and tuples retrievals can be captured.

3. *The number of tracking intervals lengths.*

   We observed from some of our experiments that the errors in AD are not produced consistently when the length of the tracking intervals changes. Therefore, we chose to measure the accuracy of AD when multiple tracking intervals lengths, which range between 2 and 8 hours, are used. The A-Detector ignores an anomaly related to a table or tuple if it is flagged based on one tracking intervals length only.

   The result indicates major reductions in the false-positive errors produced by deep inspection as a result of using multiple tracking intervals lengths. On the other hand, we did not observe reductions in the false-positive errors produced by preliminary inspection. In both cases, we observed no increase in the FNRs.

4. *Percentile (p_{upper}) used in computing training thresholds.*

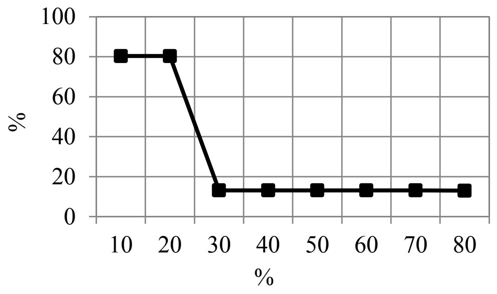
   Figure 6.3(e) compares the FPRs produced by preliminary inspection when the value of p_{upper} is set to 75% and 95%. Using a higher value for p_{upper} consistently leads to less false positive errors as it enables for higher values for the training thresholds. This result also applies to deep inspection as shown in Figure 6.4(d).

6.4.2 False-Negatives

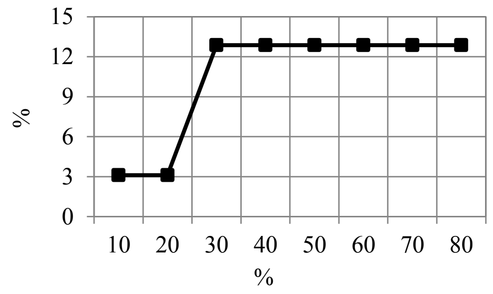
Since the query logs contain no anomalies, we replicated the evaluation queries to add anomalies to the original log in order to measure the FNRs. We refer to the number of times a query is replicated as the query replication multiple (QRM).

Based on the results of our experiments for measuring the false-positive errors, we chose the default value of the length of the tracking intervals to be equal to 6 hours. This length is guaranteed to produce low FPRs; we can thus rule out the effect of the choice of the length of the tracking intervals on the FNRs.

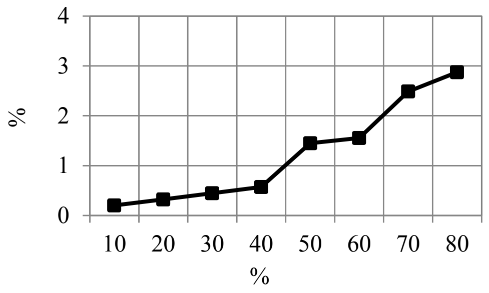
For each replicated query or group of replicated queries that lie within the same tracking interval, preliminary inspection is expected to flag an anomaly for each table referenced in the queries. However, in case of deep inspection, replicated queries that lie within the same time bucket are not expected to produce anomalies due to the use of the resolution concept. Replicating a query and setting its execution time-stamp to be at least one bucket away from the original query is expected to produce anomalies within one tracking interval of the original query. Our approach to measuring the FNRs is thus to consider each tuple retrieved by the replicated queries individually



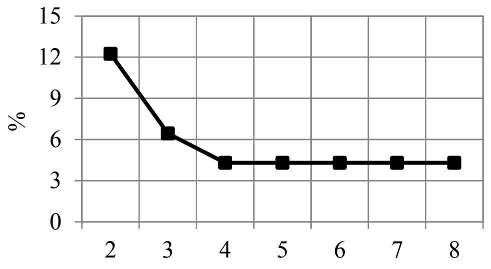
(a) FPR (measured in terms of the ratio between queries flagged as anomalous and the total number of evaluation queries) vs. size of the training data



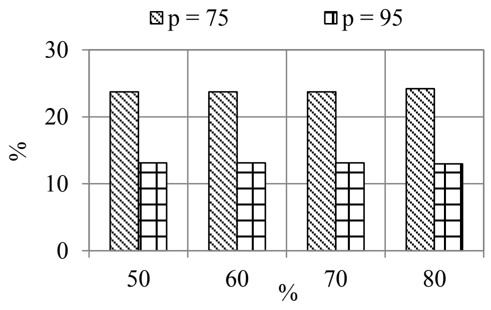
(b) Average number of tables referenced per hour vs. size of the training data



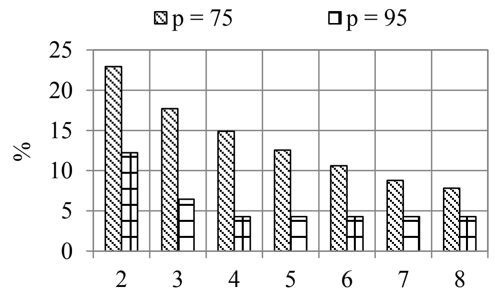
(c) Average number of tuples retrieved per hour vs. size of the training data



(d) FPR vs. length of the tracking intervals



(e) FPR vs. size of the training data for different values of p_{upper}



(f) FPR vs. length of the tracking intervals for different values of p_{upper}

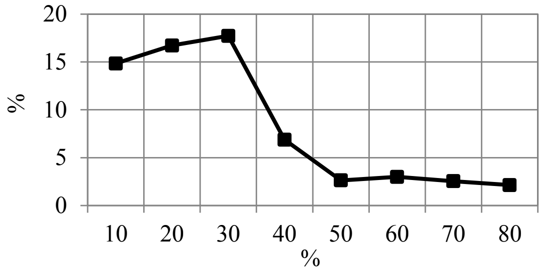
Figure 6.3.: False-positive errors by preliminary inspection

rather than considering queries that are flagged as anomalous. If no anomalies are flagged for one of these tuples, we count this error as one false-negative error.

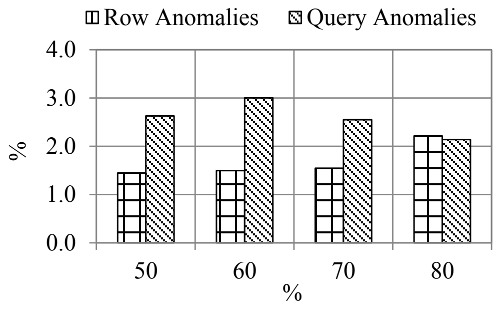
We considered the effect of the following factors on the FNRs.

1. *Number of query replications.*

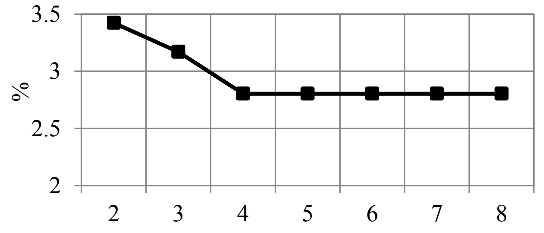
   We measured the FNRs for values of the QRM that range between 1 and 3. We set the distance between a query and its i-th replica to i * 20% the length of the



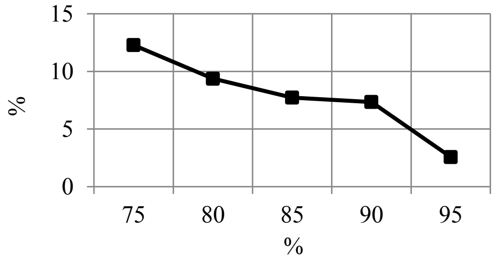
(a) FPR (measured in terms of the ratio between queries flagged as anomalous and the total number of evaluation queries) vs. size of the training data



(b) FPR vs. size of the training data



(c) FPR vs. length of the tracking intervals



(d) FPR vs. value of p_{upper}

Figure 6.4.: False-positive errors by deep inspection

tracking intervals; in this case, replicated queries will not belong to the same bucket as the original query or any other replica.

Figures 6.5(a) and 6.6(a) show the average FNRs produced by preliminary inspection and deep inspection respectively for different values of the QRM. The result indicates that the accuracy of AD increases as the number of references to the same table or tuple increases, i.e., when the thresholds are expected to be reached. The accuracy of deep inspection is lower than preliminary inspection. The FNR becomes constant for higher values of the QRM; that is because if the threshold associated with a table is met, all succeeding references to this table within the same tracking interval will be considered anomalous. In general, there is 87% probability that the first anomalous reference to a table is detected and 98.8% probability that the second anomalous retrieval of a tuple is detected.

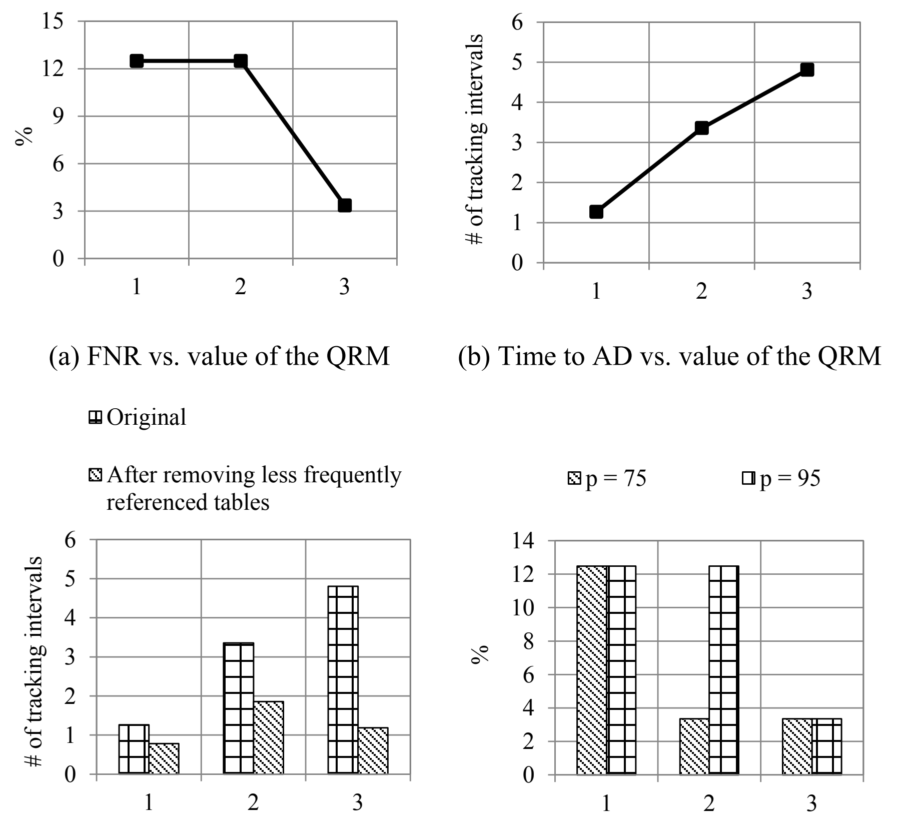
The time to AD is an important metric to consider in the evaluation. We considered two units for measuring the time to AD. The first unit is the number of anomalous references to a table or tuple that are considered normal before a related anomaly is flagged. Although it is intuitive to use this unit for measuring the time to AD, the values for this unit for references to tables cannot be easily interpreted as they depend on the training thresholds and the time of the anomalous reference within the tracking interval that includes the reference.

The second unit for measuring the time to AD resolves the problems associated with the first unit by considering the length of the tracking intervals in the computation. The time required to detect an anomaly related to a table is computed as the ratio between the number of anomalous references to the table that are considered normal before the anomaly is flagged and the threshold of reference to the table. It is to be noted that the second unit is not suitable for measuring the time to the detection of anomalies related to tuples because of two reasons: (1) the number of references to the tuples of a table during one tracking interval is different from the threshold that is related to the table and stored in the profiles because the resolution concept is used, and (2) a tuple is not as frequently referenced as a table and it is thus useful to consider each reference to a tuple rather than considering all number of references to it during a tracking interval.

Figures 6.5(b) and 6.6(b) show the time required by preliminary inspection and deep inspection to detect anomalies. The average time required to detect anomalies in references to tables is high and is equal to 5 times the length of the tracking intervals when the value of the QRM is equal to 3. When considering the individual tables for computing the time to AD, we observed that the time to the detection of anomalies related to tables that are less-referenced in the training logs is longer. We then removed the anomalies related to these tables and recomputed the time to AD. Figure 6.5(c) compares the time to AD before and after removing less frequently referenced tables. Considering the results in Figures 6.5(a) and 6.5(c), we can conclude that, on average, the third anomalous reference to a frequently-referenced table can be detected after a time interval of length equal to the length of the tracking intervals.

2. *Percentile (p_{upper}) used in computing training thresholds..*

   In contrast to FPRs, using a smaller value for p_{upper}, which leads to stricter training thresholds, results in less FNRs in case of both preliminary and deep inspection as shown in Figures 6.5(d) and 6.6(c) and in less time to AD in case of deep inspection as shown in Figure 6.6(d).



(a) FNR vs. value of the QRM

(b) Time to AD vs. value of the QRM

(c) Time to AD vs. value of the QRM

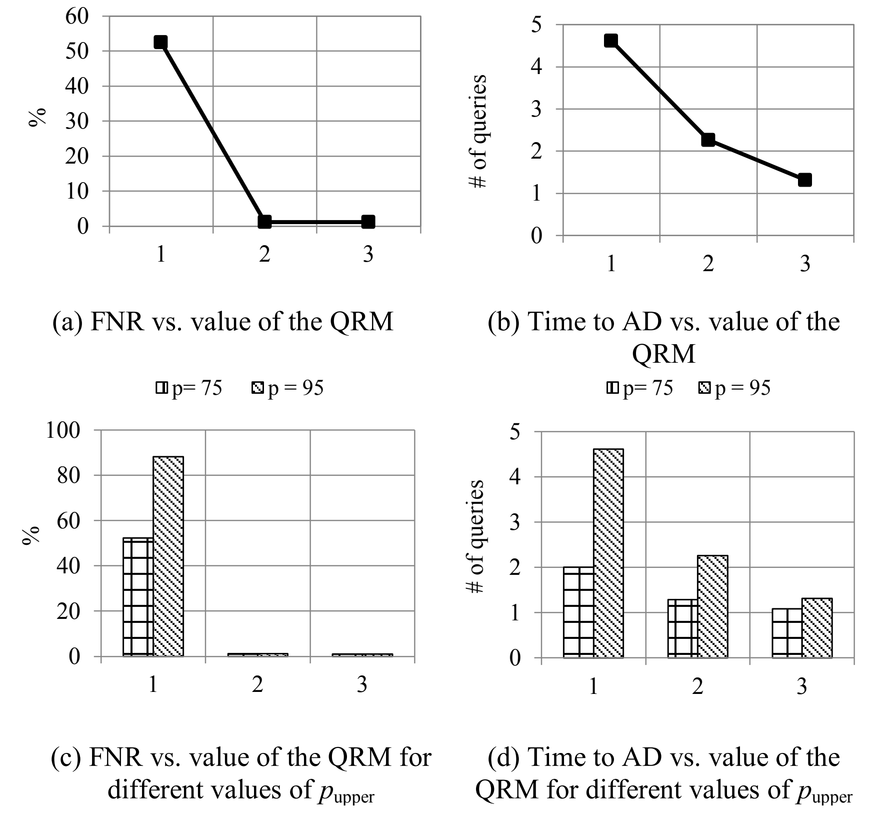
(d) FNR vs. value of the QRM for different values of p_{upper}

Figure 6.5.: False-negative errors by preliminary inspection

6.4.3 Concluding Remarks

A few remarks must be made on the results of the evaluation.

- The instability of the values of the statistical metrics described in Section 6.4.1 for small training data-sets indicates that these data-sets are insufficient for producing an adequate model. The procedure described for evaluation can thus be used before employing the proposed AD techniques to detect insufficient training data. Changes in the values of the metrics can be automatically detected using the statistical methods employed in level-shift outlier detection (LSO) [42], which is mainly used for the detection of changes (also referred to as break-points) in time-series.
- The proper length of the tracking intervals for the detection of anomalies in tables and tuples references rates is related to the lengths of user sessions. Since sessions



(a) FNR vs. value of the QRM

(b) Time to AD vs. value of the QRM

(c) FNR vs. value of the QRM for different values of p_{upper}

(d) Time to AD vs. value of the QRM for different values of p_{upper}

Figure 6.6.: False-negative errors by deep inspection

lengths are expected to vary depending on the start time of the sessions, it is important to consider the time-stamps of user queries in AD. The data-set we used for evaluation does not contain such variations; therefore, considering the time-stamps of user queries is part of our future work.

- The rates of the false-negative errors produced by preliminary inspection depend on the values of the thresholds captured during training and the variations in the rates of tables references. It is thus important to adopt other techniques for the detection of such variations. For example, one could adopt the outlier detection techniques by Kamra et al. [7] for capturing the syntactic features of normal queries and ruling out outliers in the training data for the detection of queries that are not frequently executed.
- Computing a degree of risk for each flagged anomaly is useful in eliminating false-positive errors and automating the response to detected anomalies. The degree of risk of references to tables can be easily computed based on the difference be-

tween the training thresholds and the actual users access rates. However, different approaches can be used for computing the degree of risk of anomalous tuples retrievals, e.g., the degree of risk of the individual who repeatedly retrieves multiple T-DB tuples and the degree of risk of anomalous query sequences that reference the different tables.

- Using multiple tracking intervals lengths is extremely useful in eliminating false positive errors. This approach, combined with computing risk degrees of detected anomalies that take into account multiple tracking intervals lengths, helps achieve high detection accuracy.

6.5 Conclusions

In this chapter, we presented an architecture and techniques for monitoring the rates of access to tables and tuples. The results of the experimental evaluation indicate that our techniques have low error rates only when sufficient data is available for training and the configuration parameters are selected adequately.

7 FUTURE WORK

Incomplete training profiles and using expert feedback and domain knowledge. One possible extension is the development of a Profiler that is able to detect the case of insufficient training data and flag profiles that are incomplete. A third decision, besides normal and anomalous, should also be considered by the A-Detector, which indicates when the AD decision on an action by a user who has incomplete profile is being inspected.

Towards solving the problem of insufficient training data, Costante et al. [43] propose the use of histograms to represent users profiles; an anomaly score is associated with anomalous queries and anomalous transactions based on the probabilities of the histograms bins. The main goal of using histograms is to provide *white-box* profiles that can be easily understood and edited by administrators. However, the authors did not mention how the anomaly scores computed by AD are affected by histograms modifications.

In [33], we propose using a binary classifier, which flags queries that reference one or more attributes that did not appear previously in training queries as anomalous. Profiles built by the binary classifier associate each DB attribute with a set of Boolean variables; each indicates whether the users of one role previously referenced the attribute. DBSAFE provides editable profiles by allowing administrators to toggle the values of the Boolean variables. However, the experimental results showed that the binary classifier has poor performance in practice.

Kamra et al. [7] propose using a feedback loop that changes the statistical profiles used by the naive Bayesian classifier according to AD decisions. This approach is only useful in reinforcing the AD decision and takes long time to take further effect.

The system by Valeur et al. [19] passes through an intermediate stage between the training and detection phases, referred to as the thresholds learning phase. During this phase, the training models are evaluated by selecting some queries from the training logs and computing the result of the inspection of these queries based on the current models. If some queries cannot be evaluated, the result of the evaluation will include alarms that indicate that one or more models are incomplete.

The tool by Mazzawi et al. [15] allows the user of the tool to provide feedback on the detected anomalies to indicate the correctness and importance of alerting on related actions. Possible types of user feedback on an alert related to a user action are:

1. filter-before, which indicates that the system should ignore the alert,
2. filter-after, which indicates that the alert is not important at the current time, but should be computed in case it is of some interest later and because it may affect other internal calculations, and
3. alert, which indicates that the alert is a correctly identified attack and the system should always alert on such action.

The problem of the detection of insufficient training data, monitoring user behavior and taking into account user feedback is still open for more research. One approach for incorporating user feedback that seems promising is to use active learning techniques. Active learning is concerned about the detection of incompleteness in profiles, producing minimal number of questions to be answered by human experts, and actively updating users profiles based on experts inputs.

Selection of parts of the training logs that represent the current users access patterns. Using the complete DB logs for building access profiles is inadequate in case the log represents long time intervals of queries during which one or more seasonal changes may have occurred. An initialization step is thus required by all methods that rely on training logs to create users profiles; during this step, portions of the training logs that are representative of the current access patterns of the users are selected and later used in building profiles.

Maintaining up-to-date profiles. Developing techniques for the detection of changes in the users access patterns and applying the necessary updates to the profiles during the detection phase is useful for maintaining up-to-date profiles and ensuring accurate AD. Efficient solutions are required to:

- Involve minimal human intervention,

- Provide high availability of the AD system by minimizing the maintenance time, and
- Allow for transitioning policies that can be applied during the time interval between the detection of the requirement for updating profiles until the new profiles are ready to use.

Changes to the profiles can be detected by monitoring the anomaly generation rates and looking for level shifts in data access frequencies.

Monitoring application programs. The DetAnom approach for monitoring the execution of queries by application programs is promising and can be extended to tackle the following problems.

- Taking into account DB constraints that control the program's flow, e.g., considering loops controlled by the size of the result-set of a query.
- Considering types of applications other than the standard Java desktop applications, e.g., web applications that receive inputs in the form of GET and POST requests.

Automatically inferring the values of the AD configuration parameters. Carefully choosing the system configuration parameters is important for the correct operation of AD. Leaving this task entirely to the administrators is usually inadequate and may lead to poor performance of the AD techniques. One approach for selecting configuration parameters is the use of cross-validation as suggested by Valeur et al. in [19].

Studying the effect of monitoring users connections. AD systems that monitor access to commercial DBs usually use an SQL proxy to tap the connections between the users and the monitored DB system. The impact of using the proxy on the response times to queries has not been thoroughly studied in the context of AD in DB systems. However, the study is important as the proxy may cause long delays due to reading network packets and composing queries based on packets data.

Evaluating the impact of integrating the periodicity monitoring algorithms to systems that detect anomalous non-periodic queries. A periodic query that is not detected by the profiler will lead to the generation of false-positive alarms during the detection phase if the query is not commonly executed by the issuer or the users of his/her role. The analysis of correlations between queries that are found to be anomalous is thus useful in the detection of missed periodicities.

Supporting slight changes in periodic queries features such as the syntactic features and selectivity information. Such changes occur as a result of user modifications to the syntax of the periodic queries and changes to the data of the monitored DB, which lead to out-dated data statistics.

Developing a technique for tracking the number of connection attempts by each user. Such a technique is useful in preventing a malicious insider, who is aware of the limitations on the aggregate selectivities imposed by the security system proposed in Section 5, from disconnecting from the T-DBMS to clear the tracking data-structures.

Considering the sessions' start times in the evaluation of sessions. This feature is also useful because it directly impacts the values of other features considered in SE+, i.e., the aggregate selectivities and sessions' lengths.

REFERENCES

[1] George Silowash, Dawn Cappelli, Andrew Moore, Randall Trzeciak, Timothy Shimeall, and Lori Flynn. Common sense guide to mitigating insider threats. Technical Report CMU/SEI-2012-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=34017, accessed 08/22/2016.

[2] Holger Schulze. Insider threat spotlight report. Technical report, Information Security Community on LinkedIn, 2016. http://www.infosecbuddy.com/wp-content/uploads/2016/07/Insider-Threat-Report-2016.pdf.

[3] Elisa Bertino. *Data Protection from Insider Threats. Synthesis Lectures on Data Management.* Morgan and Claypool Publishers, 2012.

[4] Elisa Bertino and Gabriel Ghinita. Towards mechanisms for detection and prevention of data exfiltration by insiders: Keynote talk paper. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 10–19, New York, NY, USA, 2011. ACM.

[5] Malek Ben Salem, Shlomo Hershkop, and Salvatore J. Stolfo. *A Survey of Insider Attack Detection Research*, pages 69–90. Springer US, 2008.

[6] Software Engineering Institute. Analytic approaches to detect insider threats. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2015. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=451065, accessed 10-28-2016.

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

[8] Qingsong Yao, Aijun An, and Xiangji Huang. Finding and analyzing database user sessions. In *Proceedings of the 10th International Conference on Database Systems for Advanced Applications*, DASFAA'05, pages 851–862, Berlin, Heidelberg, 2005. Springer-Verlag.

[9] Asmaa Sallam, Qian Xiao, Elisa Bertino, and Daren Fadolalkarim. Anomaly detection techniques for database protection against insider threats. In *2016 IEEE International Conference on Information Reuse and Integration, IRI 2016, Pittsburgh, PA, USA, July 28-30*, 2016.

[10] Asmaa Sallam, Daren Fadolalkarim, Elisa Bertino, and Qian Xiao. Data and syntax centric anomaly detection for relational databases. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 6(6):231–239, 2016.

[11] Asmaa Sallam and Elisa Bertino. Detection of temporal data ex-filtration threats to relational databases. In *Proceedings of the 4th IEEE International Conference on Collaboration and Internet Computing*, CIC '18, Philadelphia, PA, USA, 2018. IEEE.

[12] Asmaa Sallam and Elisa Bertino. Detection of temporal insider threats to relational databases. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, volume 00, pages 406–415, Oct 2017.

[13] Asmaa Sallam and Elisa Bertino. Result-based detection of insider threats to relational databases. In *Proceedings of the 9th ACM Conference on Data and Application Security and Privacy*, CODASPY '19, pages 25–35. ACM, 2015.

[14] Sunu Mathew, Michalis Petropoulos, Hung Q. Ngo, and Shambhu Upadhyaya. A data-centric approach to insider attack detection in database systems. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, RAID'10, pages 382–401. Springer-Verlag, 2010.

[15] H. Mazzawi, G. Dalal, D. Rozenblatz, L. Ein-Dorx, M. Niniox, and O. Lavi. Anomaly detection in large databases using behavioral patterning. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1140–1149, April 2017.

[16] José Fonseca, Marco Vieira, and Henrique Madeira. Integrated intrusion detection in databases. In *Proceedings of the Third Latin-American Conference on Dependable Computing*, LADC'07, pages 198–211, Berlin, Heidelberg, 2007. Springer-Verlag.

[17] Lorenzo Bossi, Elisa Bertino, and Syed Hussain. A system for profiling and monitoring database access patterns by application programs for anomaly detection. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[18] Syed Rafiul Hussain, Asmaa M. Sallam, and Elisa Bertino. Detanom: Detecting anomalous database transactions by insiders. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, pages 25–35. ACM, 2015.

[19] Fredrik Valeur, Darren Mutz, and Giovanni Vigna. A learning-based approach to the detection of sql attacks. In *Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'05, pages 123–140. Springer-Verlag, 2005.

[20] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1226–1238, 2005.

[21] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[22] Everton Alvares Cherman, Jean Metz, and Maria Carolina Monard. A simple approach to incorporate label dependency in multi-label classification. In Grigori Sidorov, Arturo Hernández Aguirre, and Carlos Alberto Reyes García, editors, *Advances in Soft Computing*, pages 33–43, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[23] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13, 2007.

[24] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[25] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–136, 1982.

[26] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[27] Sanjoy Dasgupta. Performance guarantees for hierarchical clustering. In *15th Annual Conference on Computational Learning Theory*, pages 351–363. Springer, 2002.

[28] Hochbaum and Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[29] Douglas Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[30] Michail Vlachos, Philip Yu, and Vittorio Castelli. *On Periodicity Detection and Structural Periodic Similarity*, pages 449–460.

[31] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. OLTP-Bench: An extensible testbed for benchmarking relational databases. *Proc. VLDB Endow.*, 7(4):277–288, December 2013.

[32] J. Glasser and B. Lindauer. Bridging the gap: A pragmatic approach to generating insider threat data. In *2013 IEEE Security and Privacy Workshops*, pages 98–104, May 2013.

[33] Asmaa Sallam, Elisa Bertino, Syed Hussain, David Landers, Mike Lefler, and Donald Steiner. DBSAFE - An anomaly detection system to protect databases from exfiltration attempts. *IEEE Systems Journal*, 11(2):483–493, June 2017.

[34] Frank E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[35] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[36] Peter J. Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, August 1999.

[37] F. T. Liu, K. M. Ting, and Z. H. Zhou. Isolation forest. pages 413–422, Dec 2008.

[38] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[39] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[40] Renchu Guan, Xiaohu Shi, Maurizio Marchese, Chen Yang, and Yanchun Liang. Text clustering with seeds affinity propagation. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):627–637, April 2011.

[41] George Jenks. The data model concept in statistical mapping. In *International Yearbook of Cartography*, volume 7, pages 186–190. 1967.

[42] Tsay Ruey S. Outliers, level shifts, and variance changes in time series. *Journal of Forecasting*, 7(1):1–20, 1988.

[43] Elisa Costante, Sokratis Vavilis, Sandro Etalle, Jerry den Hartog, Milan Petković, and Nicola Zannone. A white-box anomaly-based framework for database leakage detection. *Journal of Information Security and Applications*, 32:27 – 46, 2017.