Insider Threat on Databases:

Modeling and Mitigation Techniques

by

GÖKHAN KUL

June 20, 2018

A dissertation submitted to the

Faculty of the Graduate School of the

State University of New York at Buffalo

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Engineering

DISSERTATION COMMITTEE

CHAIR

Dr. Shambhu J. UPADHYAYA

Dr. Oliver KENNEDY

Dr. Varun CHANDOLA

Dr. Tevfik KOSAR

A great man once said *"There is so much we do not know on the things we count ourselves experts on that would make a book."* If there is anything I've accurately proved in this dissertation, it is that guy.

To my wonderful parents, and my amazing wife, who will never flip a page of this dissertation, but will be proud of me because of it.

And, to my advisors, Dr. Shambhu Upadhyaya, Dr. Oliver Kennedy, and Dr. Varun Chandola, who guided me out of various pitfalls, and made all this possible.

I would also like to thank my collaborators Dr. Andrew Hughes, Duc Luong, Ting Xie and Gourab Mitra for their valuable contributions.

This material is based in part upon work supported by the National Science Foundation under award number CNS - 1409551, and Turkish Ministry of National Education under Law - 1416. Usual disclaimers apply. We used experiment data from work supported by National Science Foundation under award number CNS - 1629791. Our use of the data (Study #763839-1) was ruled exempt by University at Buffalo's IRB (Federal Assurance #FWA00008824).

Contents

Abstract

Insider threat is one of the most prevalent problems for organizations, including military, intelligence communities, and business enterprises. One of the crucial reasons why it is very difficult to deal with data leakage and insider attacks is the trust aspect of the legitimate actors (i.e., employees). When an employee misuses the legitimate access rights, gains unauthorized access to a resource, steals someone else's access credentials, or even unintentionally gets their access credentials stolen, they become an *insider threat*. Unauthorized account openings and money transfers, identity thefts, and credit frauds are just a few well-known examples of this attack type.

Most database security monitoring systems depend on rule-based analysis and take action against attack types such as SQL injection attacks. Nowadays, database security research is focused on outsourced database integrity and encrypted databases. These fields usually focus on preventive measures. Detecting such attacks is a complementary feature to the preventive measures.

In this dissertation, primarily, the detection of insider threats to database management systems is addressed. First, a threat model that addresses the most common insider attacks to databases is created. This model is built considering the existing most common attack types, defense models, and their complexities. Then a framework for modeling normal behavior of users on database systems for detecting anomalous behavior while considering temporal behavior drift is presented. This framework extracts data access probability distributions from user issued queries and measures the statistical distance between them to identify the drift. It also utilizes deep learning to identify perpetrators in case of an attack. Finally, the usage areas of ontologies are investigated in order to reduce the false positive rate when the system detects an anomaly.

The experiments are conducted by utilizing two large real-world datasets: query data collected from a national bank, and query logs from a smartphone-based testbed. The smartphone query logs are well annotated and used to validate the methodologies we created. The bank data, on the other hand, is heavily anonymized, and there is no user information associated with the SQL queries. Hence, the bank data is used to test the feasibility of using the methods presented in this dissertation in real-time.

The specific tasks addressed in this dissertation are building a threat model for insider threats against relational databases, analyzing the existing attack and defense models in terms of complexity, creating a framework to construct normal behavior models of users on a database, providing a defense strategy against the modeled threat, and investigating the feasibility of ontologies to improve accuracy and usability of insider threat detection systems.

Chapter 1

Introduction

Insider attacks have become an ubiquitous problem for entities ranging from individuals to large organizations and the governments. The security branches of many corporations demand sizable budgets in order to be able to detect and respond to the cyber attacks promptly [25]. Nonetheless, *insider attacks* stay as one of the most severe types of cyber attacks. They occur when employees misuse their legitimate access rights, or gain unauthorized access to a resource such as file systems, databases and servers [11]. Unauthorized account openings and transactions, identity theft, and credit fraud are just a few other well-known examples that this attack type includes.

The "2016 Global State of Information Security Survey" [82] states that in 2015, 34% of the security incidents originated from current employees while 29% of the incidents originated from former employees. Another study in 2016 [79] shows that the percentage of organizations that experienced at least one insider incident increased from 35% to 41%. A 2015 study [78] surveyed a range of cyber attack types, and determined that insider attack is the most costly attack type to an organization. Financial organizations took the hardest hit from insider attacks, costing the highest annualized worth compared to other sectors. Another study in 2014 [16] found out that although 37% of organizations in the U.S. have

experienced such an incident, and only 3% of them were reported to the authorities due to lack of evidence.

There are some upfront security policies used and enforced in organizations to protect the integrity and confidentiality of the resources. For instance, as a multi-level security (MLS) policy, *Bell-LaPadula* [9] has very rigid constraints, and does not tolerate normal changes in business needs over time. Policies like the *Chinese Wall Security Model* [13] focus on preventing conflicts of interest in a commercial domain rather than explicitly protecting resources. A more permissive approach is based on role-based access control (RBAC) [85], where predetermined user groups are permitted to perform specific tasks. However, even here, the specific tasks performed by each group might evolve over time. There are also security policies especially tailored for preventing insider attacks [81], and enforcing policies [91]. Although useful, these policies cannot prevent insider attacks as there are often policy violations and exceptions in the implementation [10]. These exceptions can be explained with the *Oracle Policy*, *Feasible Policy* and *Configuration Policy* concepts [10]. *Oracle Policy* represents the ideal case which is what the policy maker actually intends with the policy. It can be non-deterministic but it supplies a correct answer for any question asked as it covers the intent and custom. *Feasible Policy* substitutes the ideal case with an implementation of it on a computer system. Lastly, *Configuration Policy* expresses the application of the *Feasible Policy* on a particular system. This means that the ideal case expressed in the policy may not always be enforced on the computer system.

Consequently, the difficulties of dealing with insider attacks are threefold. First, we are dealing with trusted employees who have inside knowledge about the organization, security infrastructure, and information flow. Second, creating fine-grained and restrictive access policies for shared resources restrict a legitimate actor's ability to adapt to new or unexpected tasks, while permissive policies allow room for exploitation. Lastly, a new legitimate activity from a legitimate actor can be perceived as an anomaly although the

intent is benign.

Database systems are one of the most important data stores in an organization. Their structured nature allows them to be used by various applications, and creates a variety of access platforms to the data such as web applications, desktop platforms, terminal access, and web services. We focus on database systems in order to address the weaknesses of these systems against insider threats, and also to scale the scope of this dissertation research down to a manageable level.

Although there has been academic interest on insider threats research, there is little to no success in commercial products that deal with insider threats. This is due to the relatively high false positive rates of anomaly detection based systems that are proposed by academia. This raises three questions and the solutions that form the bulk of this dissertation are as follows:

1. *Is there a way to improve the existing techniques so that they can become more usable?*

   One of the first steps in detecting an insider attack on a database is exploring the database workload to determine the normal behavior and anomalies. Unfortunately, database query logs, which keep track of the query activity chronologically, are large and unwieldy, and it can be difficult for an analyst to extract broad patterns from the set of queries found therein. Clustering the workload based on the similarity of the queries is a natural first step towards understanding the workload. The accuracy of the clustering directly affects the performance of the anomaly detection system that relies on it. However, this raises the questions of *"What makes two SQL queries similar?"* and *"How can we increase clustering quality?"* We survey various SQL query feature extraction techniques to be able to compare the similarity of SQL queries, and we explored the clustering accuracy of these methods.

2. *Is it possible to create the ideal anomaly detector for mitigating insider threats?*

There are a variety of factors that should be taken into account when detecting anomalies. User intent analysis, timing of an event, normal behavior of each user, and the role of the person in an organization are just a few examples. Each of these factors needs to be taken into account individually and together to make the most accurate decision. However, it could be difficult enough to assess even one of these factors. For instance, if we want to detect the user intent, we should start with inspecting the SQL query itself. When we take a closer look at the SQL query feature extraction methods, they essentially extract *projections*, *selections*, *joins*, *group-by*, and *order-by* items from the SQL query or a subset of them to be used in query similarity comparison. The similarity metric varies based on the aim of the extraction method; sometimes, distinct queries produce the same set of features, and sometimes queries that aim to perform the same task can produce different sets of features. Of course, the results of queries also depend on the data stored in the database. Hence, a query can be perceived with varying *interpretations*, which makes it impossible to extract the definitive *intent* of the query writer from a given query. However, it is still possible to sense a fuzzy notion of the aim. We equate this fuzzy notion with the *pseudo-intent* concept in Formal Context Analysis (FCA). A *formal context* is a triple $\mathbb{K} = (\mathbb{G}, \mathbb{M}, \mathbb{I})$, where $\mathbb{G}$ is a set of *objects*, $\mathbb{M}$ is a set of *attributes*, and $\mathbb{I}$ is a *relation* that associates each object $\mathtt{g}$ with the attributes satisfied by $\mathtt{g}$. In order to express that an object $\mathtt{g}$ is in relation $\mathbb{I}$ with an attribute $\mathtt{m}$, we write $\mathtt{g}\mathbb{I}\mathtt{m}$. Identifying if a subset of attributes is a pseudo-intent is shown to be coNP-Complete [57]. We show that it requires access to a large range of information, which is usually not available, and solving a very complex problem to achieve understanding a pseudo-intent. It gets more complex as the workload and data size grow. Instead, we argue that it is more

practical to apply approximation algorithms.

3. *Are there any other factors or methods that have not been explored before?*

   Current monitoring mechanisms depend on detecting anomalies in a user's behavior, where the user behavior is captured in different ways. The basic pathway for these systems is to: (1) extract relevant features that reflect the user behavior, (2) cluster similar actions, and (3) find outlier actions, which appears to be a very effective approach. However, such solutions are likely to face a crucial shortcoming: they do not consider the *normal drift in the user behavior over time*. Ignoring this key aspect can result in high false alarm rates. For instance, in large companies and government organizations, there are usually different units to perform different functions. Individuals and autonomous systems working for these units perform a variety of tasks; some of the roles assumed in these units can be very repetitive and straightforward while some of them require complicated operations, and they can vary in nature. We investigate the behavior drift of people using a real world smartphone query log in order to validate this assumption, and we apply various red-teaming approaches to measure the effect of considering behavior drift in comparison to the traditional approach. This work yields that this approach can dramatically reduce the false positive rates in insider threat detection systems.

   We also construct a cyber-ontology that models the overview of the normal structure of a bank, employee and customer relationship to create a knowledge-base. The scheme we constructed takes a suspicion alert as input, which triggers the ontology mechanism to analyze the chronology of the events. Our model formulates the ordinary processes that take place in a financial organization. It systematically evaluates the output of the anomaly detection mechanism in a sequential order in order to determine if the event has led up to any misuse. The output of the ontology-based system

is reviewed and labeled as *benign* or *malicious* by the security personnel. These labels are used as a feedback mechanism in the system to reduce false positives.

1.1 Outline

In this dissertation, we start with introducing fundamental concepts we discuss such as databases and database security, and perform a literature survey on insider threat research in Chapter 2. We show how SQL queries are generally structured to lay a foundation of SQL query feature extraction. We then talk about database security concepts such as SQL injections, outsourced databases and encrypted databases. Lastly, we present the existing work on insider threats.

We formulate the problem of insider threats to database management systems in Chapter 3. We build the threat model we are addressing, and we perform the complexity analysis of existing strategies to deal with insider threats. We test the feasibility of using these methods in real time by running timing experiments utilizing these methods with SQL query workload collected from a national bank.

In Chapter 4, we present the outline of a defense strategy that focuses on anomaly detection by evaluating the query workload. This chapter describes how the clustering mechanism works, and how the clustered SQL query sets are used to detect anomalies.

We perform an evaluation on the clustering quality of SQL query similarity metrics in Chapter 5. We identify different SQL query similarity metrics in the literature, and measure the quality of the query clusters created using these methods. We then apply various query standardization techniques on the queries to increase the clustering quality.

In Chapter 6, we investigate the feasibility of using temporal behavior change for anomaly detection. We measure the behavior change with the change in accessed resources in the database, and use this information to determine the pace of change we expect from

each user. We compare the performance of this system with a Long Short-Term Memory (LSTM) deep learning method.

In Chapter 7, we define an ontology development strategy on misuse detection focusing on insider attacks in the financial domain. This strategy takes in anomalies detected with the methodologies described in the previous chapters, and tests the activity to find the root of the problem.

We finally conclude by providing a brief summary in Chapter 8.

Chapter 2

Preliminaries and Related Work

An insider attack is defined as "malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems [5]." This definition covers both physical and cyber attacks. Therefore, it can occur in many forms, and can be classified in different ways.

Based on how the attack is launched on information systems, insider attacks are classified in four ways:

Privilege abuse attacks. The adversaries retrieve information with their existing access permissions without needing to know to perform their duties [56]. In this attack type, the adversaries glean, or tamper information for a malicious purpose on the information or resource that they have legitimate access rights.

Masquerade attacks. The adversaries gather the credentials of a legitimate user to access the system. Using these credentials, they assume the identity of the victim, and access the information [56]. In this attack type, a user assumes the roles of another user. Even if there is a defense mechanism in place, the malicious activity of the adversaries on the resource has the credentials of the victim.

Privilege escalation attacks. The adversaries either by exploiting a vulnerability of the

system, or by social engineering, escalate their permissions to access more sensitive data which would be unavailable to them otherwise [56]. The adversaries use their own, or someone else's credentials to elevate their access rights.

Collusion attacks. The adversaries cooperate with other actors to access information that is otherwise not available to them. The other actors may or may not be aware of the malicious intent. This attack type can be launched in many different ways, is considered to be one of the most difficult to address.

Also, once an **external attacker** gains access to the credentials of a legitimate user, they can glean, or even manipulate sensitive information if the credentials have access rights. This poses a unique threat since the observed queries by that user account do not actually belong to the account owner, which corresponds to a masquerade attack. A capable external attacker can stage other attacks such as a privilege escalation attack at this point.

Based on the behavior of the attacker, they can be classified in two different ways:

Passive attacks. In this form of attacks, the adversaries attempt to observe the information channel and data sources to learn and make use of the information. However, the attackers do not attempt to affect the system resources [95].

Active attacks. This type of attacks attempt to tamper with the data, alter system resources or affect their operation [95].

Current insider threat detection mechanisms usually depend on detecting anomalies in a user's behavior, focusing on a specific type of resource, or a combination of resources [83]. These resources can be shell commands [73], file accesses [68], and SQL queries [48, 64]. Another branch focuses on psychological factors instead of resource usage [10].

The defense strategy varies based on the attack type, the behavior of the adversary, and the specific attack strategies. When setting up defense mechanisms, organizations must consider a couple of factors before employing their strategy:

Human factors. Research in this field focuses on job satisfaction, training and experience,

and unintentional mistakes [39].

Ethical. Monitoring the activities of the employees can be critical in detecting insider threats before they turn into attacks. However, this brings a lot of ethical considerations to make when implementing a defense strategy [94].

Legal. Organizations are responsible for the data they handle. In case of a leak, they should be able to perform a forensic analysis to find the extent of the leak, and make sure the vulnerability is fixed. There are many laws regulating federal and state organizations to log information access to sensitive information [94].

Cost. Large organizations allocate considerable budgets to prevent cyber attacks. However, this budget should be channeled into where it is most required by performing a risk analysis [76].

2.1 Understanding Insider Threat

The prior research in dealing with insider attacks takes two different directions. One considers psychological aspects and the other considers physical aspects of insider threats. The research approach taken, however, can be categorized as technical, social or socio-technical [47]. The phrases “insider" and “insider threats" are terms that have ambiguous definitions, but are known to many for what they mean. Most of the research which has been done on insider threats is mainly on the psychological structure and incentives of these attacks and how to prevent them on general cases. There are cases in which researchers have focused on physical threats from insiders. For example, the work performed by Szefer *et al.* [98] focus on how to protect data centers from physical attacks and insider threats. Some instances may include a recently terminated employee, a user on a computer that is logged in, or even a janitor. No matter who the insider is, the potential threat to an organization is a problem that many organizations need to account for.

Hunker and Probst [46] go into detailing what exactly an insider and insider threat are, while giving examples of solutions to the problem of insider threats. They give definitions for “insider", “insider threat", as well as detailed issues that arise when managing insider threats and the lack of data on the topic while describing the multiple approaches to an insider attack. They describe the technical and socio-technical approaches to dealing with insider attacks and discuss the sociological, psychological, and organizational approaches to dealing with insider attacks. They explore the wide range of different people that can be insiders and they describe all of the aspects that go into making someone an insider. This level of detail is carried into the description of an insider attack, showing how there can be different types including accidental threats as well as malicious attacks.

As for the technical branch, there are two main approaches to deal with insider attacks and data leakage from databases: (1) misuse detection, and (2) anomaly detection.

Misuse detection aims to collect a dataset of events that leads to intrusions. These systems observe user behavior, and when certain actions are taken by any user, the system either raises an alarm, or blocks the user from taking any other actions. These particular actions can be designed for specific scenarios to prevent well-known attacks, or they can be learned from other sources such as successfully caught incidents [56].

Anomaly detection approach, on the other hand, depends on detecting anomalies at a user’s behavior. The systems implemented with these approach can focus on a specific type of resource, or combination of resources [83] such as file access patterns [68], online and social behavior [37], command-line statements [67, 73], and SQL queries [48, 64, 97, 101].

Mathew *et al.* [66] state that insider attacks pose a serious threat due to the fact that current security systems are aimed at prevention of unauthorized access. They focus on the fact that not only can threats come from trusted entities within an organization, but a successful attack may be the result of multiple entities working together, termed insider collusion. Therefore, this is said to justify a call for monitoring and detection methods

which take into account these potential interactions between entities. From here, they go on to detail the use of a new system, called Information-Centric Modeler and Auditor Program (ICMAP), which generates Capability Acquisition Graphs (CAGs) to represent information about physical locations of data, difficulty of access to components of the data system, etc. This graph allows for feasible analysis of which paths to insider abuse targets are the least difficult to traverse. The CAG holds information about the potential difficulty of accessing certain nodes in the system, and can therefore determine the path of least resistance. This can allow for security analysts to bolster the defenses of the systems along that path, or simply to monitor activity along these nodes for suspicious behavior. It is noted that the cost of creating, updating, and analyzing a CAG is considerably high and thus impractical to maintain in real-time. The proposed solution is to only update the CAG periodically (termed “CAG milestones"), as well as search for paths vulnerable to attack using a greedy algorithm that may not give the absolute most vulnerable path in the system, but is likely to after a number of runs. They provide an example of a situation in which a collusion attack could be carried out undetected, with malicious activity performed under the guise of being legitimate work tasks. Therefore, such a scenario would be difficult to catch in the act. However, a CAG generated by ICMAP can trace the means through which somebody with only public access could obtain information with top-secret security restrictions.

2.2 Data Leakage Detection on Databases

The basic idea behind our system is to profile normal user behavior, detect suspicious behavior using this information, and distinguish malicious behavior from benign intents [56]. Indeed, this idea is not new; there are many anomaly detection systems focusing on suspicious behavior of users. As the basic unit of interaction between a database and its users, the sequence of SQL queries that a user issues effectively models the user’s behavior.

2.2.1 Databases and SQL Queries

A *relational database* is a set of relations (*i.e., tables*), and a *relation* is a bag or set of tuples (*i.e., rows*) where a tuple is a structured data item. The structure is defined with attributes (*i.e., columns*) and the types of the attributes. The difference between a bag and a set is that bag semantics allow duplicate records whereas set semantics enforce uniqueness.

Relational Database Management Systems (RDMS) ensure result accuracy while providing efficiency for joining combination of multiple resources. The most important principle in these systems is that a query simply must not return a wrong result.

Structured Query Language (SQL) is a declarative language that is designed for managing, manipulating, and retrieving from relational databases. Other than schema manipulation and data access control operations, SQL queries mainly perform 4 different operations: (1) insert, (2) update, (3) delete, and (4) select. The basic structure of these operations are given respectively:

```
(1) INSERT INTO table (column1, column2, ...)
                VALUES (value1, value2, ...);
(2) UPDATE table
    SET column1 = integer|decimal|string|...
    WHERE column2 = integer|decimal|string|...
(3) DELETE FROM table
    WHERE column1 = integer|decimal|string|...
(4) SELECT [aggregation] column1, column2, ...
    FROM table1, table2
    [WHERE table1.column1 = table2.column3]
    [ORDER BY column1]
    [GROUP BY column1]
    [LIMIT integer]
```

where the brackets show optional query items. As can be implied from these basic structures, queries that perform similar tasks usually have analogous structures, or at least share some attributes.

SQL query statements are constructed from *clauses*. Every line of the query structures given above constitutes a clause. As an example let's take the following query:

```
(1) SELECT u.username, u.yearenrolled
(2) FROM user u, accounts a
(3) WHERE u.id = a.userid AND a.balance > 1000
(4) GROUP BY u.yearenrolled
(5) ORDER BY u.yearenrolled
```

Line 1 consists of the `SELECT` keyword, and the *projection* items. Line 2 has the `FROM` clause which lists the tables the query is going to use. Line 3 is named the `WHERE` clause. Where clause contains *selection* and *join* expressions. `u.id = a.userid` expression is a join expression, and `a.balance > 1000` is a selection expression. Line 4 and 5 include the *group-by* and *order-by* items, respectively.

2.2.2 Database Security

The current research on database security focuses on different security aspects. For instance, the framework we present in this dissertation aims to detect deviations from original behavior, but it cannot be used to detect the integrity of an outsourced database, or it cannot detect if an adversary takes a snapshot of the database server. However, it can be supported with, and integrated into the following database security measures.

SQL injection prevention. If an application is vulnerable to the SQL Command Injection Attacks, the adversary can gain unauthorized access to the data that they normally are not authorized to access. One approach to overcome these attacks is to randomize the keywords

in the SQL preprogrammed query [12] while the other approach is to use benign candidate inputs to mine programmer intentions [8].

Outsourced database integrity. Nowadays, many organizations prefer outsourcing the management of their systems and databases in order to reduce their costs. However, in this case, the outsourced database is accessible to the outsourced company systems and personnel which raises concerns on the query result trustworthiness. The current research focuses on interactive proofs and authenticated datastructures [106, 107].

Encrypted databases. If intruders can infiltrate into the organization network, they can take a snapshot of the database systems. This line of research focuses on running SQL queries over encrypted data against data leakage [80, 99], hence even if a snapshot attack is successful, the data stolen would be encrypted.

2.2.3 Data Leakage

There has been extensive level of research in detecting data leakage from databases, but there are still challenges in this field [86]. Chung *et al.* [24] proposed the use of access patterns to databases to detect typical behavior of users. Kamra *et al.* [48] developed a SQL query feature extraction method that generalizes complex queries into simpler, and easier to compare forms to use them in detecting insider attacks. Mathew *et al.* [64] introduced a data–centric approach that requires access to the data that a query returns, which then would be used to make the similarity of the query computation via computing the overlap between returned result sets. Gafny *et al.* [35] proposed a similar framework to Mathew *et al.*, adding the application context information on top of the detection mechanism, such as location and user type. Costante *et al.* [27] presented a scheme to find the root cause of data leakage in databases, and identifying the severity of the leak. Wang *et al.* [102] focused on harvesting attacks considering query correlation and result coverage.

2.3 Temporal Concept Drift

Although temporal concept drift has not been studied in the insider threat detection and prevention domain, there have been research on distinguishing the normal changes in the behavior and the anomalies since it can be used to reduce high false positive rates for many applications.

Maggi *et al.* [62] is one of the leading works that introduced concept drift in web applications. Their model is designed to track the changes on websites in order to find out if there is a need to retrain the security application.

One of the early works that introduced the temporal aspect in database workloads proposed that learning timing of different types of queries by specific users could improve intrusion detection rate [59]. This addresses queries that are issued out of working hours, or irregular updates to the database which usually take place regularly.

Our approach on the other hand, requires adaptation to individual behaviors which would allow flexibility of adaptation to new tasks and duties, hence reducing false positive rate without compromising true positive rate.

2.4 Ontologies on Insider Threats

Mundie *et al.* takes the initiative to create an ontology framework for insider threat research [70]. They focus on standardization of the terms insider and insider threat while investigating the relationships between these terms. Their main aim is to provide a better understanding of the conceptual model of the insider threat, hence eliminating the inter-study differential and facilitating a standardization of terms.

In their paper, Costa *et al.* [26] detail the creation of an ontology for use in describing the indicators of insider threats. The primary reason cited for focusing on this area is that

it had been uncommon for information about these insider threats to be circulated outside of the businesses that were typically subject to them; without a standardized method to abstract the data, doing so would have meant releasing confidential data related to the attack. Without public circulation of this information, progress in determining methods to prevent these insider threats has been severely hampered despite increasing focus in this area of research. The ontology was developed with the aid of over 800 cases of malicious insider activity compiled from various sources, all of which were natural language descriptions of the incident. These cases were analyzed using a semi-automated method which had output relationships between common concepts which were used as the basis of the classes for the ontology. The top-level classes used were “Actor, Action, Asset, Event, and Information." The ontology can then describe scenarios by showing the relationships between subclasses of these top-level classes. The paper goes on to give a series of examples for how to use the ontology to further the field of insider threat detection. While it starts by restating the usefulness of this level of abstraction for publicizing information related to threats without also disclosing organization-sensitive information, the paper also goes on to note that the semi-automation of data collection that the ontology implementation paves the way for others to develop detectors for indicators of insider threats. Also, the paper states that it would be possible for this work to be extended such that event logs (and other operational data) as well as information that organizations keep about insiders that is not as a result of direct interaction with information technology (human-resources data) could be translated and parsed in order to automatically create ontology individuals. If these processes are automated, then this would make it possible for a semantic reasoner to be constructed to classify insiders as instances of subclasses within the ontology, which would provide a clear view of specific indicators of threats. Ultimately, the development of this ontology appears to be a valuable stepping stone to further progress in the area of insider threat detection, but its greatest benefits will be lost if it is not widely used in a standard form.

The ontology schemes proposed in [26, 70] differ from the methodology presented in Chapter 7. Our methodology aims to ensure the integration with other knowledge domains to enable data integration and it models the financial domain while including a basic overview model of insider threat. The ontology modeled can be used to systematically evaluate any insider threat detection schemes in a realistic way and discover attacks that share similarities with previously identified attacks. However, this scheme assumes that there is historical data on insider attacks caught.

2.4.1 Summary

In the first part of this chapter, we discussed the types of insider attacks, and how they can emerge. We then surveyed the literature on insider threat while discussing how understanding the threat can help dealing with them. Second, we focused on data leakage from databases. We introduced fundamental concepts such as databases and database security, and performed a literature survey on insider threat research. We showed how SQL queries are generally structured to lay a foundation of SQL query feature extraction. We then talked about database security concepts such as SQL injections, outsourced databases and encrypted databases. Third, we presented how concept drift is utilized in anomaly detection in the literature. Lastly, we investigated how ontologies are used in insider threat research.

In the next chapter, we investigate the modeling aspects of insider threat which is the starting point for the development of insider threat mitigation schemes.

Chapter 3

Formal Threat Model and Complexity of Insider Attacks to Databases

Insider attacks are one of the most dangerous threats to an organization. Unfortunately, they are very difficult to foresee, detect, and defend against due to the trust and responsibilities placed on the employees. In this chapter, we first define the notion of user intent, and construct a model for the most common threat scenario used in the literature that poses a very high risk for sensitive data stored in the organization's database. We show that the complexity of identifying pseudo-intents of a user is coNP-Complete in this domain, and launching a *harvester* insider attack within the boundaries of the defined threat model takes linear time while a *targeted* threat model is an NP-Complete problem. We also discuss about the general defense mechanisms against the modeled threats, and show that countering against the harvester insider attack model takes quadratic time while countering against the targeted insider attack model can take linear to quadratic time depending on the strategy chosen. We analyze the adversarial behavior, and show that launching an attack with minimum risk is also an NP-Complete problem. Finally, we perform timing experiments with the defense mechanisms on SQL query workloads collected from a national bank to

test the feasibility of using these systems in real time.

This chapter is organized as follows: We start by mathematically formulating the intent model in Section 3.1. We then describe the threat model in Section 3.2. We discuss other factors that can be utilized in both threat and defense models in Section 3.4. In Section 3.3, we perform running time experiments on real-world data to test the feasibility of using the defense mechanisms in real time. Finally, we summarize in Section 3.5.

3.1 Intent Model

The SQL queries that a user issues on a database could model the normal usage behavior [28]. Also, queries that are similar in nature imply that they might be issued to perform similar duties [54]. However, understanding the intent of the query is regarded as being as hard as constructing a new query, and it gets even more complex when the query is complicated [36]. For this reason, capturing the intent of a query is often ambiguous [4], and there can be various feature extraction methods. In the literature, SQL feature extraction to capture the intent of a query has been studied for different purposes such as performance optimization [6], workload analysis [4, 63], query recommendation [22, 51, 104], and security purposes [48]. As the need to access more complex information, the construction of the query intrinsically gets complicated, which often makes it troublesome to understand the resulting query for human readers. It gets harder to compare the similarities of the queries in terms of accuracy when the complexity of the question increases, even though they are created to accomplish the same task [4].

When we take a closer look at these feature extraction methods, they essentially extract *projections*, *selections*, *joins*, *group-by*, and *order-by* items from the SQL query, or a subset of them, to be used in query similarity comparison. The similarity metric varies based on the aim of the extraction method; sometimes distinct queries produce the same set of

features, and sometimes queries that aim to perform the same task can produce different sets of features. Of course, the results of queries also depend on the data stored in the database. For example, the queries

```
SELECT * FROM user WHERE username LIKE "A%"
SELECT * FROM user
```

will produce exactly the same result if all the `username` values in the user table start with "A." Hence, a query can be perceived with varying *interpretations*, which makes it impossible to extract the definitive *intent* of the query writer from a given query, but it is still possible to sense a fuzzy notion of the aim.

We equate this fuzzy notion with the *pseudo-intent* concept in Formal Context Analysis (FCA). A *formal context* is a triple $\mathbb{K} = (\mathbb{G}, \mathbb{M}, \mathbb{I})$, where $\mathbb{G}$ is a set of *objects*, $\mathbb{M}$ is a set of *attributes*, and $\mathbb{I}$ is a *relation* that associates each object $\mathtt{g}$ with the attributes satisfied by $\mathtt{g}$. In order to express that an object $\mathtt{g}$ is in relation $\mathbb{I}$ with an attribute $\mathtt{m}$, we write $\mathtt{g}\mathbb{I}\mathtt{m}$ [7].

Identifying if a subset of attributes is a pseudo-intent is shown to be coNP-Complete [7]. It is also important to note that not all sets of attributes in a context represent a pseudo-intent; counting the number of pseudo-intents is proven to be #P-hard, while finding the number of sets that are not pseudo-intents is shown to be #P-Complete [31].

Thus, to understand the pseudo-intent, we extract the relevant features from SQL queries, and following the definition provided by FCA, we define the attributes as the SQL query features. These attributes are essentially the resources consumed by the SQL query. We will refer to these resources when we say *query intent* – (*pseudo-intent in FCA domain*), from now on. We take a closer look into the SQL query feature extraction and their quality in Chapter 5.

Definition 1 (Query intent)**.** *Under the assumption that resources consumed by a user to perform a task reflect the user's intent, we define intent as a finite bag of resources denoted*

by $\phi = \{r_1, r_2, ..., r_{|\phi|}\}$.

This definition directly conforms with the feature extraction methods described above where r_i is the extracted query features.

Definition 2 (User Activity)**.** *User activity A is represented by a user $u \in U$, where U is the set of all users, for the time period T that starts from t_0 and goes on for Δt, and the set of intents ϕ performed by u within T. Formally,*

$$A_u^T = (a_u^{t_0}(\phi), a_u^{t_1}(\phi), ..., a_u^{t_n}(\phi)) \tag{3.1}$$

where $a_u^{t_i}$ represents a timestamp of an activity performed by user u.

The users usually create similar workloads on the database to perform their daily tasks [54]. These workloads can be utilized to create a chain of tasks for each user.

Definition 3 (User Activity Graph)**.** *User activity graph G is a directed and weighted graph that shows the historical navigation of the user between resources. $G = (V, E)$ where V is the set of all possible activities performed by the user and E is the set of navigations from one activity to the other.*

There can be two types of weights: (1) the number of different items between two sets, and (2) cumulative probability of the next activity where the probability is calculated by the division of the number of occurrences of an item to the total item occurrences.

3.2 Threat Model

Stallings [95] classifies security attacks as *passive* and *active*, and defines them as follows: "*A passive attack attempts to learn or make use of information from the system but does not*

affect system resources. An active attack attempts to alter system resources or affect their operation."

In this chapter, we survey the research focusing on detecting data leakage by insiders using query logs. The state-of-the-art of this research targets *passive attackers*: this type of attackers query the database to extract sensitive information. The threat model assumes that the insider does not tamper with the data, or execution on the client-side application. They may access the database system through a client-side application, or through direct interaction with the database server while still having the queries observed by the query monitor.

We do not address *active attackers* who tamper with the data, or query results because we aim to address the threat posed by insiders who try to steal information, in order to prevent identity theft, and information leakage. To address data tampering by an insider, the construction can be supported with integrity verification techniques [49], and authenticated data structures [107].

Our system does not block such behavior, but it observes the query activity of all the users, and alerts the security personnel if it catches a suspicious activity.

However, our system does not address *snapshot attacks* in which the adversary copies the database server, or the database instance completely. Such attacks cannot be caught by the query monitor since it doesn't have any control on the server instance. The cloned server instance, if the attack is successful, would be available for use of the adversary. In that case, the adversary wouldn't need to go through the query access monitor to query the database.

Our model is specifically effective against preventing insider attacks to databases.

We consider two passive attack models; harvester (*a.k.a. aggregate*) and targeted (*a.k.a. individual*) attackers [71]. The harvester attacker is an adversary who manages to replicate one part of the database for exploratory analysis by querying the database. The targeted

attacker is, on the other hand, an adversary who accesses certain information without needing to know for legitimate purposes, but chooses the attack parameters carefully to avoid being detected.

Unlike an attacker who is trying to access information from outside, an insider usually has knowledge about the database and procedures within the organization. Using this knowledge, they can exploit the privileges and trust placed on them, and access information that they are not supposed to see.

If an adversary can issue a query and not get detected by the defense mechanism, the query evaluation on just one table takes linear time. Consider a query Q_1 issued on a database D. To find the tuple t that the user requests with the selection clause $\sigma_{s=c}(R) = t$ where s is the attribute name in table R and c is the constant the query is searching for, the database system scans through all the values on $R.s$, which is performed in $O(|R|)$ time. Of course, this can be made faster; for example if there is an index built on the column previously, the evaluation of Q_1 can be improved up to $O(log|R|)$ time. However, as the intent gets more complex, the evaluation takes more time. It is known that query evaluation problem is NP–Complete for conjunctive queries [18], and the space complexity of query evaluation problem for relational algebra, in general, is PSPACE–Complete [52].

3.2.1 System Architecture

The query monitoring system (QMS) is designed to be modular and flexible, in order to be able to work with other security applications. It can be integrated into any DBMS just to observe the query traffic so that it doesn't bring timing overhead to the query processing. After it is active and running, any application that interacts with the database can be monitored.

When an application user uses a web or desktop application on their computer (client-

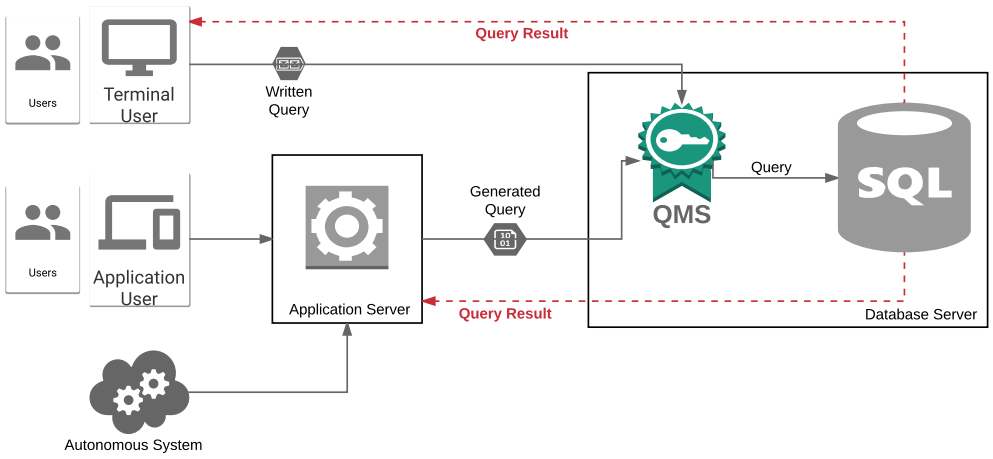


Figure 3.1: An architecture of query monitoring

side), the application generates queries, and issues them to the database. On the other hand, the terminal users interact with the database via a terminal interface on their computers, which directly connects them to the database server. The database is contained in a database server (server-side). QMS just observes the queries that are issued to the database, and it doesn't block or change the queries. Any query that is issued to the database is captured by the QMS, processed, and logged there, and then sent to the database. Although the system does not block any queries, it detects suspicious activity, and reports them to the security personnel. The overview of the system architecture is depicted in Figure 3.1 where QMS acts as the observer in the system.

Next, we define the threat model this construction is built against.

3.2.2 Analysis of Harvester Attacks

Harvester attacks focus on collecting a variety of information from the database, which result in search behaviors that show high levels of *diversity* and *broadness* according to the state-of-the-art solution presented by Wang *et al.* [103]. Diversity is measured by the query and parameter similarity within a database session while broadness is measured by

the variety of the return results from the queries within a database session. In the rest of this section, we discuss the threat, and defense models based on the approach presented earlier.

The harvester threat model

We assume that the adversaries within the organization have the domain knowledge about the database schema, or have access to software tools that run on the database without requiring the user to have familiarity to the underlying database, but they don't have any insight into the data content stored in the database. When the attackers are not looking for specific information, they can issue exploratory queries on the system. This can be in two forms: (1) none or few filtering conditions, or (2) a lot of queries with filtering conditions. Since this attack type does not have a specific target information, the adversary has only one goal: extracting as much information as possible. In the most basic form, the adversary would issue wildcard queries for each table, and can retrieve all the database by issuing number of tables $|R|$ queries.

Consider a database `MY_BANK` with relations `CUSTOMER` which includes details about the customers of a bank, `ACCOUNT` which contains all the accounts the bank is handling, with many-to-many relationship, and `CUSTOMER_ACCOUNTS` which has the information of which accounts belong to which customers as shown in Figure 3.2. An insider who has information about `MY_BANK` schema can access all the data in the database with only 3 queries: `SELECT * FROM customer`, `SELECT * FROM account`, and `SELECT * FROM customer_accounts`.

It is also possible for the adversaries to utilize filtering conditions, which would result in increasing the number of queries, to avoid using wildcard queries to protect themselves from detection. However, the query monitor, in the end, would log the resources accessed while evaluating the query and would eventually end up logging the same resources in both

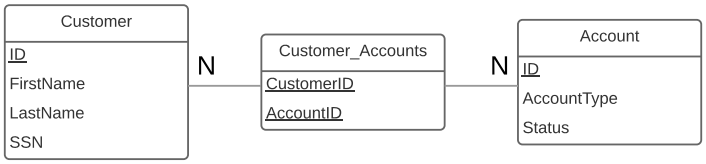


Figure 3.2: MY_BANK database diagram

cases. Hence, although there can be a lot of ways to harvest all the data, the complexity would still be $O(|R|)$.

The harvester defense model

Analyzing queries mostly relies on the structure of queries [48] since access to the data in the DBMS may not always be possible for auditing systems, or people responsible for investigating attacks. Systems that utilize query correlation usually exploit the resources the queries want to access as mentioned in Section 3.1, and, hence, the features in the intent set can be used to measure diversity. Data-centric query comparison [64], on the other hand, requires access to the data, and can be time consuming since it involves query evaluation. After the query evaluation phase, the search results of the queries should be compared to measure broadness.

For instance, Makiyama *et al.* [63] approach query log analysis with a motivation of analyzing the workload on the system. They extract the query terms in selection, joins, projection, from, group-by, and order-by items separately, and record their appearance frequency for each query in the dataset. They create a feature vector using the frequency of these terms, which they use to calculate the pairwise similarity of queries with cosine distance function. The feature extraction method presented in this work can be used to measure diversity [48]. As we indicated before, for the logging mechanism, there is no difference between using wildcard queries, and using the resource names directly in the

query. For example:

```
Q1: SELECT * FROM customer
Q2: SELECT ID, FirstName, LastName, SSN FROM customer
```

When the query parser receives the query $Q1$, it processes the $\star$ as `ID`, `FirstName`, `LastName`, and `SSN`. Hence, the query similarity function used would consider $Q1$ and $Q2$ as the same and $\phi_{Q1} = \phi_{Q2} = \{ID, FirstName, LastName, SSN\}$.

The complexity of feature extraction from the query can be considered $O(1)$. To create the intent for all queries ϕ_{ALL} in a user's session, we should process all the queries in that session. Hence, the time complexity to create the intent set is $O(n)$, where n is the number of queries issued. To compute the diversity, we compare the resources in ϕ_{ALL} with all the resources possible (S) in the database. The time complexity of this operation is $O(|\phi_{ALL}| \cdot |S|)$.

3.2.3 Analysis of Targeted Attacks

Targeted attacks focus on specific information in the database, and investigate for sensitive information that the attacker is interested in. The activities can be similar to the common, and legitimate activities of the user. However, the attacker acts without needing to know for any legitimate purposes [48, 54, 97].

The targeted threat model

Any user, including an insider, should consider not only the data retrieved including the needed data, but also should consider retrieving only the most important results when issuing the query to the database. Knowing that issuing too many queries on a database could raise suspicions, a crafty intruder should be precise about the information that they need. Thus, the retrieved rows should provide the maximum coverage, and minimum re-

dundancy. Still, the question remains if the resources that the insider wants to access are in the database, and if they can be retrieved with accessing a limited number of resources. The motivation for this limitation can vary; the needed time to answer a query can dramatically increase as the complexity of the query increases [63], the access policy can prevent certain information from being used together [2], or accessing too much information can raise an alert [34]. Hence, the insider should be precise while preparing each expression, and avoid redundancies.

Considering that the expressions included in the query reflect the intent of the attacker, we can use our intent model presented in Section 3.1 as a base while formulating the problem:

Question 1. *(INTENT SET PROBLEM) Every time a user issues a query, given a limitation of maximum number of resources that can be accessed, does there exist a query construction that will return the information that the user is looking for on the system?*

Claim 1. *The construction of the SQL query with the user's intent is NP–Complete.*

Proof. Let V_1 be the feature set that includes all possible columns and constants. Let V_2 be the user intent set of an attacker. Let k be the maximum number of resources that can be accessed. We can construct a graph $G = (V, E)$ where $V = V_1 \cup V_2$. Note that $V_1 \cap V_2 = \emptyset$. For any $u \in V_1$ and $v \in V_2$, we include the edge $(u, v) \in E$ if the expression v includes the item u. An intent set is a set $I \subseteq V_1$ such that every node in V_2 is a neighbor of at least one element of I; namely, for each $v \in V_2$, there exists $u \in I$ where $(u, v) \in E$.

The problem of *Intent Set* is to determine for a tuple (V_1, V_2, E, k) if the graph $G = (V_1 \cup V_2, E)$ contains an intent set of size at most k. *Intent Set* is in NP. Given a tuple (V_1, V_2, E, k) and a candidate intent set I, the verification of it being an intent set can be performed by checking if $I \subseteq V_1$, each element in V_2 has a neighbor I. This operation can be performed in polynomial time.

We can show Intent Set is **NP–Complete** by reducing Set Cover, a known NP–Complete problem, to Intent Set. Let (U, S, k) be an instance of Set Cover, where U is a set of n elements, S is a collection of m subsets of U, and k is some value $1 \leq k \leq m$. Set Cover asks if there exists a group of k or fewer sets from S such that their union equals U. Let $V_1' = S$, $V_2' = U$, and (s, u) is in E' if $u \in s$. Then (V_1', V_2', E', k) is an instance of Intent Set. If $G' = (V_1' \cup V_2', E')$ has an intent set of size k or less, it directly corresponds to a covering of U of size k or less. If G' does not have an intent set of size k or less, then U cannot have a covering of size k or less. Therefore, Set Cover reduces to Intent Set in polynomial time.

As Intent Set belongs to NP and Set Cover is NP–Complete [44] and reduces in polynomial time to Intent Set, Intent Set is NP–Complete. □

Hence, given the relation between the user's intent and the resources required for each intent, the problem to determine if a set of k resources provides access to the desired intent is NP-Complete. The general intuition this proof conveys is that, without having knowledge about the contents of the database, and allowing access to only k number of resources, a query should be constructed very carefully by an attacker to avoid risk of detection. However, as the user intent grows, the restriction k imposes will make it more challenging, possibly intractable, to construct a query that provides access to the target data.

Example. Non-admin users

For illustration purposes, let's consider Ashley, an adversary. She decides to gather sensitive information of some users who are not administrators for a masquerade attack. To find such data, Ashley needs to issue a query that filters the rank of the users, and brings sensitive information about them that she can use. Let's assume that the current credentials that Ashley uses limit her to access at most 4 resources.

There are many factors that she has to consider: "*What credentials can be useful?*", "What resources can be used?", "Does the data she is looking for exist in the database?"

and so on. She can come up with a query such as `SELECT username, rank FROM user WHERE rank <> "admin"`.

When we apply this method to the query `SELECT username, rank FROM user WHERE rank <> "admin"` as shown in Figure 3.3, the intent set we need to use is

$$I = \{username, rank, <>, "admin"\}$$

Consequently, she would be using 4 resources, and the query given would provide usernames of non-admin users. However, if the credentials she uses do not have access rights to at least 4 resources, she won't be able to issue this query without getting caught.

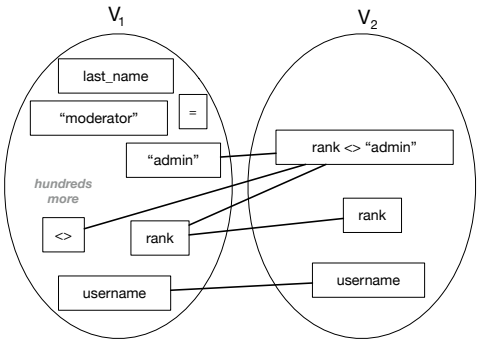


Figure 3.3: Intent Set for `SELECT username, rank FROM user WHERE rank <> "admin"`

Approximation. According to Claim 1, Intent Set is NP–Complete. However, there is a greedy algorithm that gets an approximation ratio of *ln n*, with loosened constraints. Consider that the attacker has access to a benign query set (i.e., query log). In that case, the attacker can pick the query that covers the most points in V_2. The attacker throws out all the points that the picked query covered, and repeats.

Claim 2. *If the optimal SQL query construction uses k sets, the greedy algorithm finds a solution with at most $k * lnn$ sets.*

Algorithm 1 Greedy Intent Set Algorithm

procedure GREEDY-INTENT-SET(V_1, V_2, k)
    $U \leftarrow V_2$
    $C \leftarrow emptyset$
    while $r \neq 0$ AND $k * lnn \geq \mid C \mid$ **do**
        Select an $S \in V_1$ that maximizes $\mid S \cap U \mid$
        $U \leftarrow U - S$
        $C \leftarrow C \cup \{S\}$
    end while
    return C
end procedure

Proof. The optimal query construction uses k sets. Consequently, there are some sets that cover at least $1/k$ of all the points in V_2. If we choose a query that covers the most points in V_2, it will cover $1/k$ points or more. This leaves $n * (1 - 1/k)$ points in the intent set. When we repeat the same operation m times, it leaves $n * (1 - 1/k)^m$ points. As a result, in at most $m = k * lnn$ rounds, our operation will complete. If $P \neq NP$, this is the best approximation ratio we can expect. □

The implication of this algorithm is that, the attackers who have read access to query logs can use this information to construct their attack strategy. Following this approximation algorithm, the attackers may stop at a point where they are satisfied with the information they accessed without needing to construct the optimal SQL query.

The targeted defense model

The distinctive traits of targeted attacks are that the adversaries access information without needing to know, and deviate from their normal behavior. There are two approaches to mitigate these attacks: (1) misuse detection, and (2) anomaly detection. Misuse detection focuses on detecting predetermined specific malicious activities. The defense mechanism creates rule sets on what kind of behavior a user should not perform which requires fine-grained security analysis on the system, and identifying what information each user should

not have access to. Anomaly detection focuses on deviations from normal patterns.

Misuse detection mechanism is a rule based system, which checks every activity of a user to determine if it matches with the forbidden behavior defined in the rule set. This check is usually straight-forward, and takes linear time.

Anomaly detection mechanism, on the other hand, forms *user profiles* in order to formulate a normal behavior pattern for each user. We take every query issued by all users and extract the resources consumed to use them to create the user profiles. After that, we can approach this problem as a clustering problem. The information collected is labeled with a clustering algorithm like k-means, or hierarchical clustering, while keeping the resource-user association. A user profile can be created from the distribution of labels to each cluster. Utilizing this information, the anomaly detection mechanism observes each query issued by each user, and catches the anomaly if a user's distribution starts to shift from the profiled distribution [48, 97].

Clustering based anomaly detection systems require a one-time clustering operation to create profiles. This operation can be repeated to update the user profiles as time progresses. The time complexity of the clustering operation depends on the selected technique; if it is a technique that requires a pairwise distance matrix, the operation has quadratic complexity, and if it is a heuristic based technique like k-means, the operation has linear complexity [17].

3.2.4 Analysis of Adversarial Behavior

The adversaries launch an attack when they are convinced that the time and effort spent, and the risk taken to reach the information are worth the value of the information [87]. We assume that the adversary, being an insider, knows that there are security measures in place against insider attacks. They can research on the defense strategies, and inquire about

the system beforehand to find out the weaknesses in order to reduce the risk of exposure. However, while trying to reduce the risk, the adversaries still need to gain access to some *key information* to reach their target.

In this section, we describe a model called *key risk graph* that shows different paths that the adversaries can take to reach their target. We can naively describe every step in this path as follows, and as represented in Figure 3.4:

- State representing the adversary's current status, S_1
- State representing the adversary's desired status, S_2
- Key information needed to go from S_1 to S_2, key_i
- Risk associated with the move if the adversary has key_i, r_1
- Risk associated with the move if the adversary does not have key_i, r_2

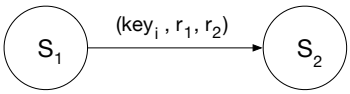


Figure 3.4: Key risk representation

Definition 4 (Key Risk Graph)**.** *Key Risk Graph KG is a directed graph that shows the adversary's informational states, key information needed to change the current state to another, and risks associated with going one state to another.*

$KG = (V, E; K, V_0, V_S, \pi, \delta)$ where V represents the set of all possible states, and E represents the set of navigations from one state to the another. K is the set of key information, V_0 is the starting state, V_S is all the other states that can be traversed only if the risk associated with one of the incoming edges is taken by the adversary, $\pi : V \to K$ is a function that assigns keys to the states, and lastly, $\delta : E \to K \times \mathbb{N} \times \mathbb{N}$ is a function that assigns risks to the edges where $\mathbb{N}$ represents all natural numbers.

Given a Key Risk Graph, the problem of finding an attack with the minimum risk is **NP-Complete** since there is a direct mapping from KEYSEQ problem [23], where the authors provide a proof with a reduction from 3–SAT to KEYSEQ.

To further illustrate how the Key Risk Graph can be utilized in practice, consider a plausible real-life scenario as below.

Example. Identity Theft and Credit Fraud

Actors: Alice (Adversary), Bob (Victim), Celia (Victim), Dan (Collaborator)

Bob, the customer, goes to a bank to payout his auto-loan. While explaining why he is there, he mentions that he and his wife sold a house that they just inherited, and they are paying their debts using that money. Alice, the banker, figures the rest of the money must be in his wife's account. After completing the transaction, Bob leaves the bank. Alice decides that if she can find out the SSN number and birth date of Bob's wife, she can use that information to apply for credit collaborating with her contact, Dan, in the credit department. Then pay off the credit using the money in the account. There are two ways that she can access that information:

- *Querying the home address information Bob provided, she finds out there are two people living in the same address: Bob and Celia.*
- *Querying the auto-loan information, she finds out Bob's co-signer is Celia, who carries the same last name.*

Alice determines Celia must be his wife, so she looks up Celia's account information including her birth date and SSN, and passes the information to Dan.

In the example given above, Alice tackles two challenges to reach the key information she needs: Finding out who Bob's wife is, and using the information she found, she reaches her target. Assuming that she knows how the defense mechanism described in this chapter

works, she evaluates both strategies. The first strategy involves looking up an address which is usually not a common practice, hence, it is associated with a higher risk (r_2) of triggering the security system. The second strategy, checking the auto-loan information (key_i), on the other hand, is just a very common procedure while paying off a loan, hence, she finds it less risky (r_1).

3.3 Experiments

We run our experiments on a SQL query dataset collected from a national bank to show the feasibility of the defense system we discussed in Section 3.2. We are specifically interested in the timing property of the log classifier. We want to show that the clustering operation performs well enough to be practical for use in a corporate setting. As shown in the experiments, the system can cope very well with large data sizes even on commodity desktop computers.

The data we use to run our experiments is based on SQL query logs that capture all query activity on the majority of databases at a major US bank over a period of approximately 19 hours. Logs are anonymized by replacing all constants with hash values generated by SHA-256, and manually vetted for safety. Table 3.1 shows the summary of our dataset. There are 2,818,719 parsable queries in our dataset. Since we only inspect the data access patterns, we base our analysis on the 1.35 million syntactically valid `SELECT` queries.

The bank query log starts at about 11 AM. To test scalability, we vary the number of queries processed by truncating the log after varying lengths; After 1.5 hours, the log contains 250,000 SELECT queries with 756 skeletons, just over 500,000 queries after 3.3 hours, and so forth. Full statistics for each truncated log are shown in Table 3.2. The query workload is shown in Figure 3.5.

Type	Queries
Select	1,349,861
Union	1,306
Insert	1,173,140
Update	288,098
Delete	6,314
TOTAL	2,818,719

Table 3.1: Distribution of query types in bank dataset

Time	Total Time	SELECTs	Skeletons
12:34 PM	1.5 hours	250,740	756
2:24 PM	3.3 hours	512,981	946
4:06 PM	5 hours	789,050	1,057
6:27 PM	7.3 hours	972,511	1,102
9:23 PM	10.3 hours	1,079,427	1,130
12:43 AM	13.6 hours	1,176,582	1,256
04:04 AM	17 hours	1,270,984	1,576
6:28 AM	19.4 hours	1,349,861	1,614

Table 3.2: Summary of query log size by absolute time

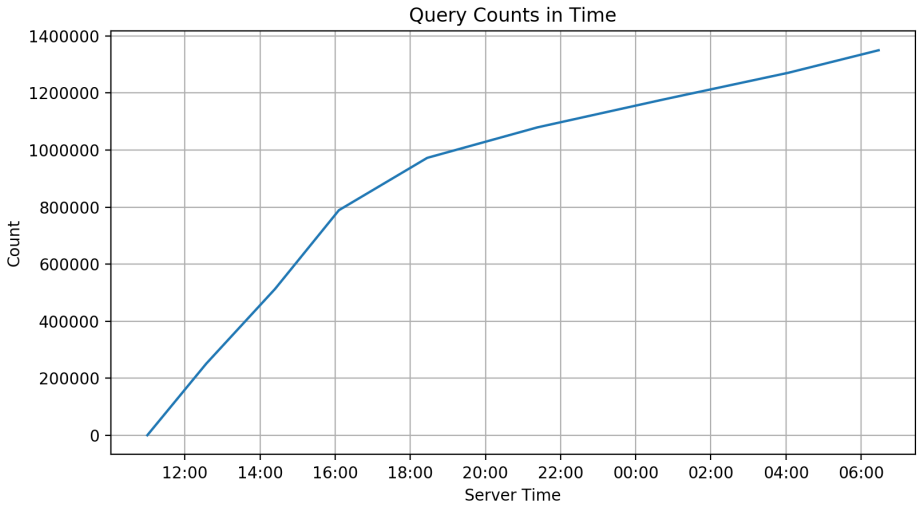
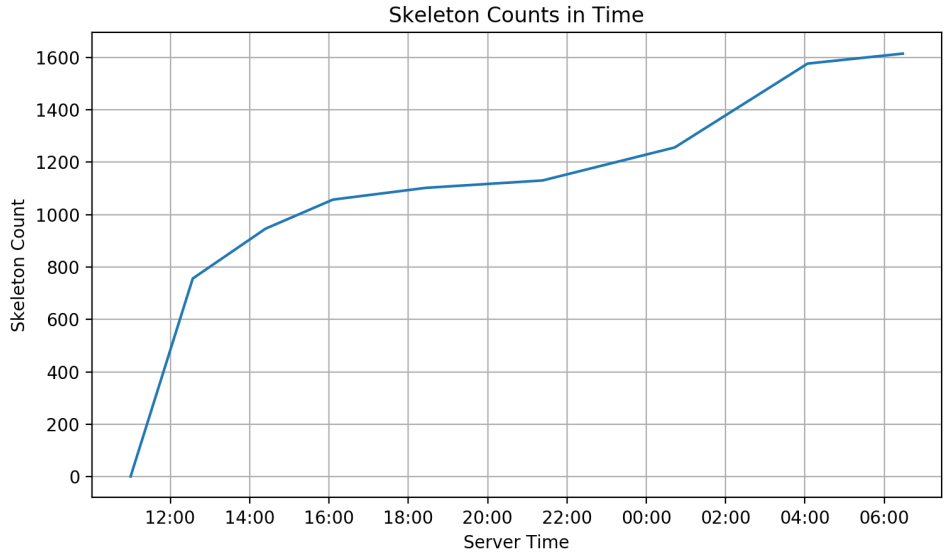


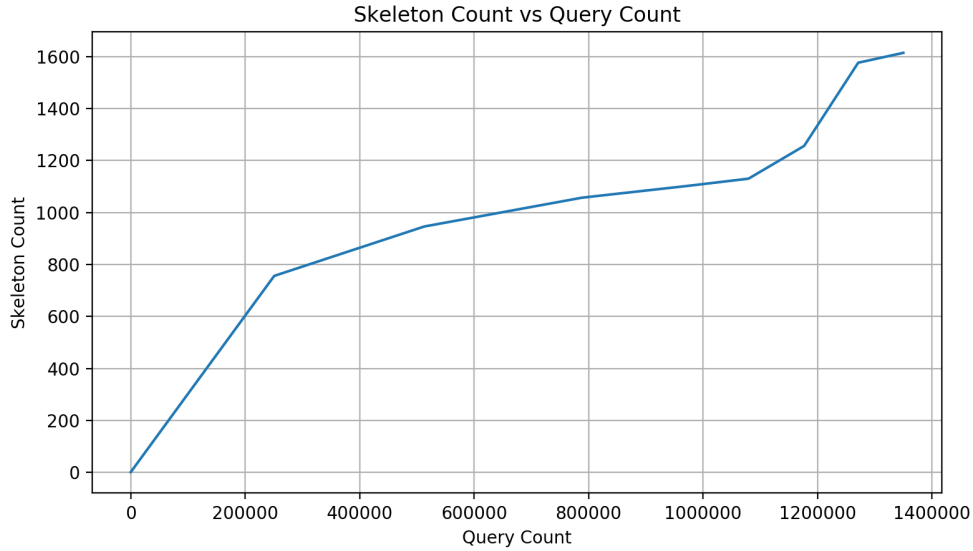
Figure 3.5: Server Query Workload

A query with its constant values replaced by a placeholder grammar term is called a *query skeleton*. Our first observation is that, there are only 1,614 different query skeletons. This means that the number of distinct structures that we need to cluster is quite small.

Furthermore, this number grows sub-linearly over time. As shown in Figure 3.6a, nearly half of all the skeletons arise in the first 1.5 hours of the trace. As the query log size grows, these queries are converted to skeletons to be clustered.



(a) Skeleton Counts in Time



(b) Skeleton vs Query Size

Figure 3.6: Skeleton Conversion

The experiments were run on a commodity computer with a 3.2 GHz Intel Core i7

processor and 16 GB RAM memory running Ubuntu 14.02. We used the R implementation of hierarchical and k-means clustering. The remaining components were implemented in Java using the single-threaded JDK 1.8.0. Reported running times are the average of 10 trials for each phase.

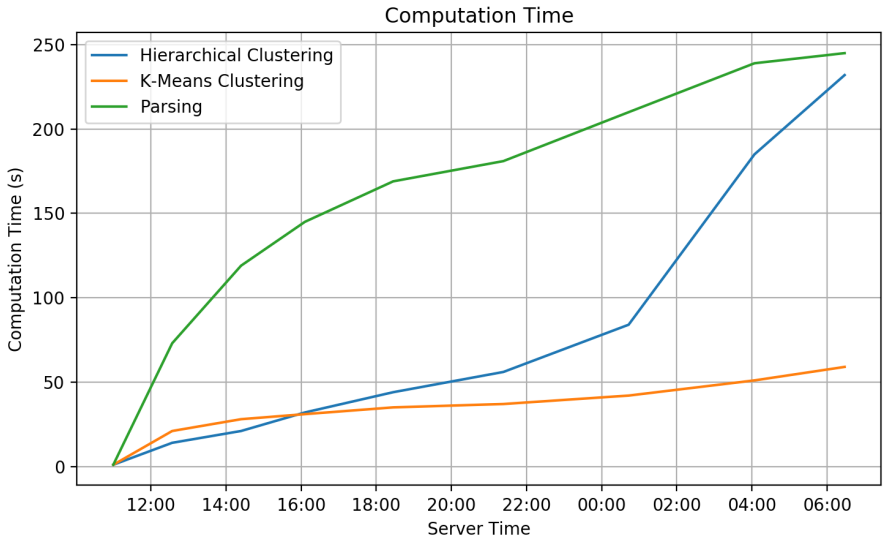


Figure 3.7: Running times for query parsing, hierarchical and k-means clustering

The running time of query parsing grows sub-linearly with the log size as can be seen in Figure 3.7. We attribute this to the dominant costs being resource allocation associated with defining and indexing new query skeletons, and potentially JIT compilation overheads. The running time of hierarchical clustering grows super-linearly with the number of query skeletons. This is typical for clustering processes in general, which need to compute and evaluate a full $O(N^2)$ pairwise distance matrix. Fortunately, the number of skeletons grows sub-linearly with the number of queries, and the overall scaling by time is roughly linear. We expect that this growth curve would eventually become sub-linear for longer traces.

3.4 Discussion

We focused on modeling the insider threat to the databases while analyzing the computational complexity of the attacks, and defense mechanisms.

Our work paves the way for creating approximation approaches for computationally complex attack, and defense models analyzed in this chapter. Such approximations are not guaranteed to provide optimal results; however, a computationally feasible approximation method can make NP–Complete problems tractable. A better understanding of the complexity of the attacks will help with developing appropriate defense mechanisms.

The defense mechanisms discussed in Section 3.2 can further be improved by exploiting the sensitivity level of data accessed by the users, or by constructing a *User Activity Graph*.

One of the most obvious ways to determine the sensitivity level of data in a database is that getting every column in a table evaluated by domain experts. For example, in a bank database, `USER` table might include *FirstName*, and *LastName* columns along with *SSN*. While there is little to no risk of *FirstName* and *LastName* columns being queried, it might be a problem if *SSN* column is queried without legitimate basis.

However, while evaluating an activity to consider if it is a threat, the importance of the resource should not be the sole indicator of an attack. The workforce in the organization is designed to perform a specific duty that may require access to very sensitive data. Thus, the aim should be analyzing if the user actually requires that data to perform their task.

The user activity graph is built with the historical actions of the user, and could be used to predict the next move of the user with a method such as Hidden Markov Model. The utilization of the probability of the next action of a user can help the defense mechanism determine how likely an action will occur.

3.5 Summary

In this chapter, we first described the notion of user intent which we can utilize in insider threat detection research, and we discussed the complexity of identifying the pseudo-intent of a user. We defined two attack models: (1) harvester attacks, and (2) targeted attacks, while discussing the complexity of both attack, and defense mechanisms in the literature. We then discussed the key risk problem. Finally, we showed that the defense mechanisms can, in fact, be used in production systems in terms of running time.

In the next chapter, we present the outline of our system to deal with insider attacks.

Chapter 4

SQL Query Intent Analysis

Insider threats to databases in the financial sector have become a very serious and pervasive security problem. This chapter presents a framework to analyze access patterns to databases by clustering SQL queries issued to the database. This dissertation contributes to an insider threat detection system called Ettu, which works by grouping queries with other similarly structured queries. The small number of intent groups that result can then be efficiently labeled by human operators. We show how our system is designed and how the components of the system work. Our results show that our system accurately models user intent.

This chapter is organized as follows. We introduce our core contribution, a technique for query intent modeling and describe Ettu[1], a system that uses query intent modeling as a way to flag potential insider attacks in Section 4.1. In Section 4.2 we evaluate the feasibility of query intent modeling, a concept introduced in Chapter 3, and we summarize the contents of this chapter in Section 4.3.

[1] Ettu is derived from the last words of the Roman emperor Julius Caesar, "*Et tu, Brute?*" in Latin, meaning "*You, too, Brutus?*" in English to emphasize that this system is meant to detect the unexpected betrayals of trusted people.

4.1 System Outline

Ettu operates in three stages: (1) An offline **clustering** phase where query logs are aggregated and summarized so that they can be easily examined, (2) A semi-automated **labeling** phase to identify potential signs of insider attacks, and (3) A **pattern-generation** phase that creates pattern matchers that screen queries online as they are processed by the database. These stages are illustrated in Figure 4.1.

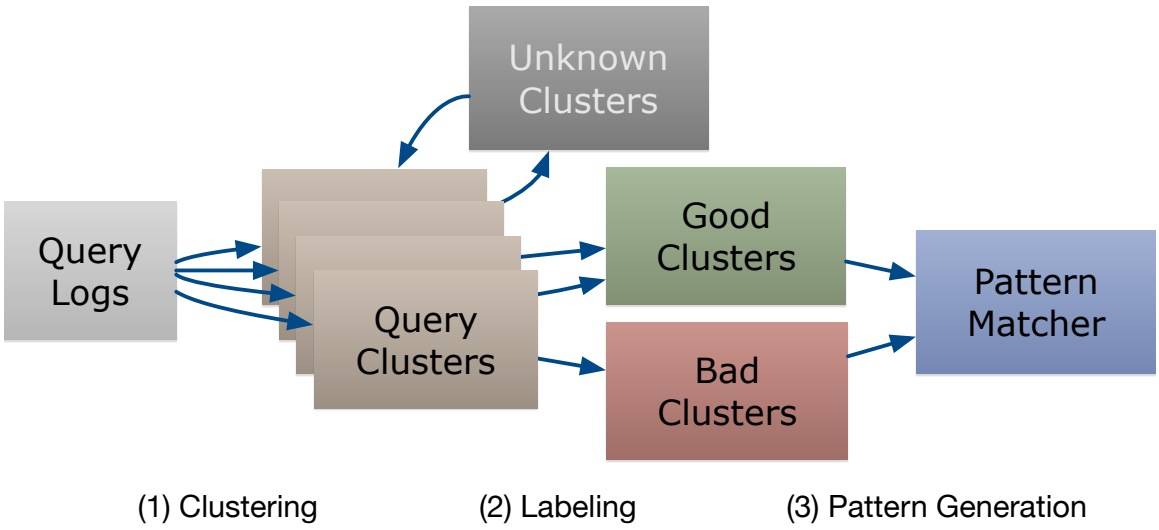


Figure 4.1: The typical Ettu workflow.

The initial input to Ettu is a log of query activity processed by the target database. The log is annotated with supplemental metadata like usernames and timestamps. The goal of the first phase is to produce a concise, but precise summary of the log. To summarize the log, queries are clustered together into "similar" groups. Each group consists of a set of queries that are issued with similar goals in mind due to the similarity of their structures.

As a declarative language, the abstract syntax tree (AST) of a SQL statement acts as a proxy for the intent of the query author. Intuitively, our approach is based on the assumption that overlap between the ASTs of two queries implies overlapping intents. Thus, naively, we would group a query Q with other queries that have nearly (or completely) the same AST as Q. For the remainder of this chapter, we will use queries Q to denote both the

query itself and its AST encoding. This structural definition of intent has seen substantial use already, particularly in the translation of natural language queries into SQL [61].

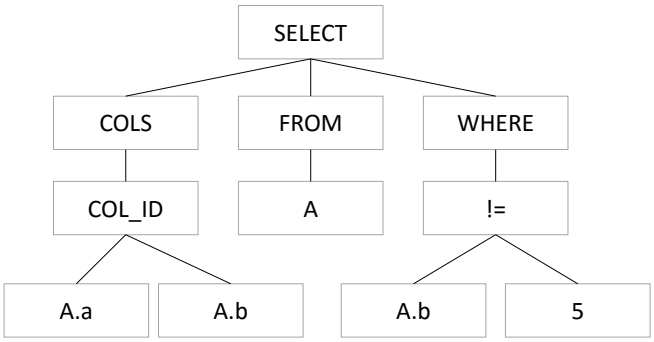


Figure 4.2: An abstract syntax tree of query `SELECT A.a, A.b FROM A WHERE A.b != 5`.

Queries are grouped by intent using a simple clustering process: (1) A query is first encoded through its AST and then summarized as a vector of *features* that identify meaningful subgraphs of the AST; (2) A distance metric defined over the feature vectors creates a measure of similarity between queries and enables standard vector-based clustering algorithms to organize queries into *intent groups*; (3) A simple user interface uses ground truth from human analysts to validate and refine the intent groups.

4.1.1 Weisfeiler-Lehman

An ideal distance metric would measure the level of similarity or overlap between ASTs and their substructures. Naively, for two SQL queries Q_1 and Q_2, one satisfactory metric might be to count the number of connected subgraphs of Q_1 that are isomorphic to a subgraph of Q_2. Subgraph isomorphism is NP-complete, but a computationally tractable simplification of this metric can be found in the Weisfeiler-Lehman (WL) Algorithm [93]. Instead of comparing all possible subgraphs of Q_1 against all possible subgraphs of Q_2, the WL algorithm restricts itself to specific subgraphs.

Given a query Q, let $N \in Q$ denote a node in Q. N is labeled with the SQL grammar

symbol that N represents. The *i-descendant* tree of N: $desc(N, i)$ is the sub-tree rooted at N, including all descendants of N in Q up to and including a depth of i.

Example 1. *Given the tree in Figure 4.2,* $desc(\texttt{COLS}, 2)$ *is the tree containing the nodes* `COLS`*,* `COL_ID`*,* `A.a`*, and* `A.b`*.*

The WL algorithm identifies a query Q by all possible i-descendant trees that can be generated from Q:

$$id(Q) = \{\ desc(N, i) \mid N \in Q \wedge i \in [0, depth(Q)]\ \}$$

Here $depth(Q)$ is the maximum distance from the root of Q to a leaf. As an optimization, subtrees are deterministically assigned a unique integer identifier, and the query is described by the bag of Q's i-descendant tree identifiers. Thus two query trees with an isomorphic subtree will both include the same identifier in their description. The WL process for the tree in Figure 4.2 is given in Figure 4.3. The bag of identifiers is encoded as a (sparse) feature vector and allows Euclidean distance to measure the similarity (or rather dis-similarity) of two queries.

The WL algorithm assumes zero knowledge about structural features of the trees it compares, limiting itself to i-dependent subtrees. Conversely, the grammar of SQL has a very well defined structure. In Ettu, we exploit this structure to eliminate redundancy and create features that more reliably encode the query's semantics. Concretely, we consider three improvements over WL. First, the number of features created by the WL algorithm is large. Although clustering naturally prunes out features without discriminative power, we can use SQL's semantics to identify structures that are unlikely to be useful. For example, the subtree $desc(\texttt{SELECT}, 1)$ in Figure 4.2 is common to virtually all queries. Such features can be pruned preemptively.

Second, the precise structure of a feature may not be relevant to the intent of the query.

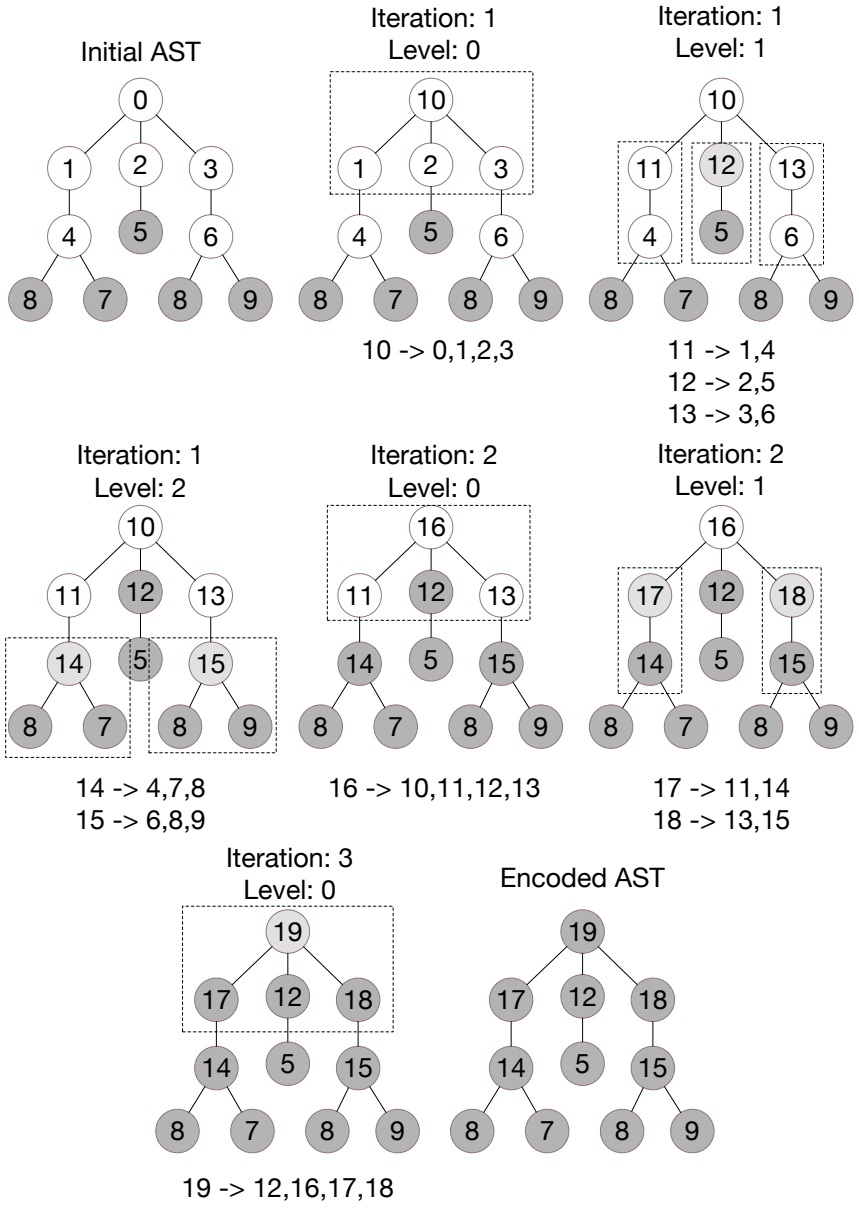


Figure 4.3: Weisfeiler-Lehman algorithm applied on AST given in Figure 4.2.

Queries that are procedurally generated often include placeholders or components that are dynamically constructed; At best, such components serve to modify a simpler, more general query intent.

Example 2. *Consider the query* `SELECT * FROM R WHERE R.a = 5`*. Although the*

subtree `R.a = 1` *identifies the specific goal of the query, the same goal could also be abstracted as* `R.a = ?` *where* `?` *denotes a placeholder constant.*

Finally, SQL makes frequent use of commutative and associative operators. The semantics of such operators may overlap, even if their i-descendant subtrees do not.

Example 3. *Consider the Boolean expressions* `A AND B` *and* `A AND B AND C`*. The former AST is a* `AND` *node with 2 children while the latter has 3. Although the two ASTs have 3 0-descendant subtrees in common, they share no 1-descendant subtrees; WL ignores the similarity between the two conjunctive expressions.*

We address these challenges below in Sections 4.1.2 and 4.1.3.

4.1.2 Query Skeletons

Given a set of queries, instead of considering all of them, we only consider the differences in their structures assuming that differences in constant values have a minor effect on the intent of queries. As mentioned earlier, a query with its constant values replaced by a placeholder grammar term is called a *query skeleton*. Hence, the two queries "`SELECT A.a FROM A WHERE A.b = 5`" and "`SELECT A.a FROM A WHERE A.b = 2`" share the same skeleton because they have exactly the same query structure except for constant values.

Analysis using the set of query skeletons instead of original SQL queries will reduce the number of distinct queries processed and in general will produce similar clusters. However, there are some cases where the constants do play a role. For example, consider the earlier example of a bank teller in Buffalo withdrawing money for a customer from California. To handle these cases, we can pass the constants themselves as additional features for the clustering algorithm.

4.1.3 Dynamic Features

Second, we generalize the WL algorithm to enable more flexible, structure-aware query tree labeling. Suppose we are given a query Q. Each node $N \in Q$ has a set of *labels* $\texttt{labels}(N)$, initialized to the singleton set containing the SQL grammar atom of N. A *labeling rule* is applied to the labels of a node N and those of its children and generates new labels to be added to $\texttt{labels}(N)$. Given a set of rules, we apply them bottom-up to the nodes of Q to compute a full set of labels for Q's nodes. Finally, the feature vector of Q is defined by the bag $\biguplus_{N \in Q} \texttt{labels}(N)$ of all labels on all nodes of Q.

Example 4. *The AST for the Boolean formula* `A=a OR B=b`*, would begin with the feature set* $\{|$`A`, `=`, `a`, `OR`, `B`, `=`, `b`$|\}$*. We could define a rule to be applied to = nodes that constructs a skeleton label: Applied to the subtree* `B=b`*, this rule would add a new label denoting the tree* `B=?`*, where* `?` *is a placeholder value.*

4.1.4 Clustering

Finally, vector-based clustering is performed on the feature vectors obtained from query skeletons. These clusters are manually classified into three categories by the user: (1) Safe Clusters that correspond to normal activities, (2) Unsafe Clusters that could potentially be insider attacks, (3) Unknown Clusters that represent too broad a group of queries to classify as safe or unsafe. Clusters of this third type are subdivided further until a set of clusters is obtained that are purely safe or unsafe.

To help administrators to reliably label intent clusters, we need a way to compactly present the set of user queries in each cluster. We adopt Frequent Pattern Trees (FP Trees) [42] for this purpose. Normally, FP Trees help to mine frequent patterns of item-sets, for example bags of items commonly bought together. Individual items in each item-bag are sorted by their global *frequency*, or the total number of times the item occurs in any

bag. An FP Tree is a prefix-trie built over these sorted item-bags. We build an FP Tree over the queries, using query features as items and query trees as item-bags. Recall that each feature corresponds to a subtree of the AST and there is a unique shared feature behind every node in the FP Tree. To describe a cluster, Ettu presents the user with an expandable view of the FP Tree. As the user selectively traverses the tree, Ettu merges the feature ASTs of nodes from the root to each expanded node to create a syntactically correct partial AST for visualization. FP trees provide three beneficial features for human inspection: (1) A high compression rate (ratio of total number of items consumed v.s. number of nodes generated in the tree), compactly summarizing millions of incoming queries per day; (2) A customizable visualization level, allowing users to settle on an appropriate level of detail without being overwhelmed.

If the items are sorted in the descending order of *frequency*, there are better chances that more prefix strings can be shared. This assumption is validated by experimental results in [42]. But this assumption fails when an item occurs very frequently but with no others. Thus, as an optimization we replace *frequency* by *total popularity*. The *popularity* of a feature in its bag is the number of distinct features that coexist with it. *Total popularity* is the sum of single bag *popularities*.

We compare the compression rates for FP Trees generated using both frequency and total popularity in Figure 4.4, which shows how compression rate varies with the number of queries incorporated into the tree. As the number of queries increases, the compression rate for both approaches increases super-linearly, suggesting that FP Trees are a good way to visualize large query clusters.

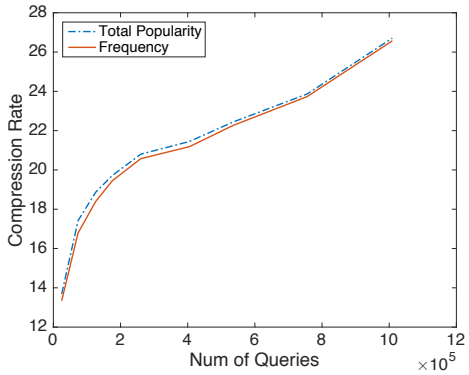


Figure 4.4: Popularity vs Frequency when determining the compression rate.

4.2 Feasibility Study

Our evaluation is based on anonymized SQL query logs that capture all query activity on the central databases of a major US bank over a period of approximately 19 hours. Although there are 1.35 million parsable SELECT queries in our dataset, there are only 1,614 different query skeletons; Most queries differ only in constants. Even this simple optimization significantly reduces the load on Ettu's users.

In order to demonstrate the feasibility of our system, we run Ettu with a sample set of 140 different SQL query skeletons, allowing us to more reliably visualize the clustering process. We perform hierarchical clustering with this query skeleton set and compare the clustering result with our manual clustering. Figure 4.5 shows correspondences between 2 trees: the dendrogram of hierarchical clustering on the left hand side and manual grouping on the right hand side. The high correlation between two trees shows that Ettu is actually able to group SQL queries based on their intents and it is possible to flag queries that deviate from normal user behavior as abnormal so that database administrator can easily inspect and prevent possibly harmful activities.

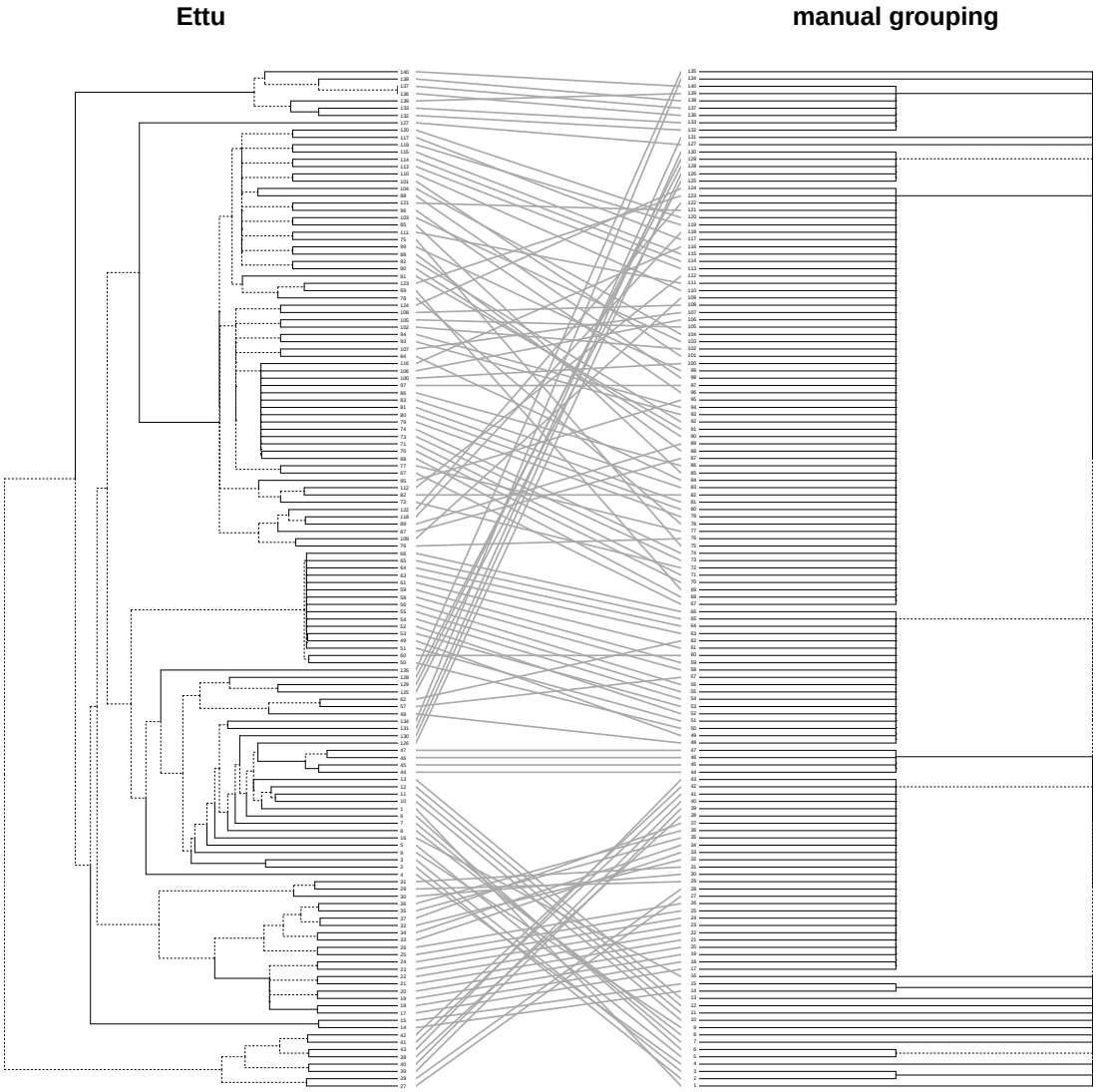


Figure 4.5: Correspondences between hierarchical clustering obtained from Ettu and from manual grouping.

4.3 Summary

In this chapter, we presented the outline of our system called *Ettu*. This system uses a defense strategy that focuses on anomaly detection by evaluating the query workload. The query workload is partitioned into clusters to distinguish structurally different queries. These clusters are labeled to identify queries specifically accessing sensitive information.

We then performed a feasibility study to measure the clustering accuracy. We concluded that the hierarchical clustering output is consistent with manual grouping, but also there is room for improvement.

In the next chapter, we investigate SQL query similarity metrics, and present a regularization method to increase clustering quality.

Chapter 5

SQL Similarity Metrics and Clustering Quality

Database access logs are the starting point for many forms of database administration, from database performance tuning, to security auditing, to benchmark design, and many more. Unfortunately, query logs are also large and unwieldy, and it can be difficult for an analyst to extract broad patterns from the set of queries found therein. Clustering is a natural first step towards understanding the massive query logs. However, many clustering methods rely on the notion of pairwise similarity, which is challenging to compute for SQL queries, especially when the underlying data and database schema is unavailable. We investigate the problem of computing similarity between queries, relying only on the query structure. We conduct a rigorous evaluation of three query similarity heuristics proposed in the literature applied to query clustering on multiple query log datasets, representing different types of query workloads. To improve the accuracy of the three heuristics, we create a generic feature engineering strategy, using classical query rewrites to standardize query structure. This strategy results in a significant improvement in the performance of all three similarity heuristics.

This chapter is organized as follows. We start by explaining the need for evaluating SQL query similarity metrics, and how they can be improved in Section 5.1. We then introduce existing techniques on log clustering and SQL query similarity in Section 5.2, and we describe a feature engineering technique called regularization in Section 5.3. In Section 5.4, we explain our query workloads and create a strategy for evaluating the quality of query similarity metrics. The evaluation is presented in Section 5.5. We discuss our experiment results, findings and ideas to further build upon the surveyed techniques in Section 5.7, and in Section 5.6, we explain how this work can be beneficial by giving real life examples. Finally, we conclude by identifying the steps needed to deploy query log clustering into practice using the techniques evaluated in this chapter in Section 5.8.

5.1 Preliminaries

Database access logs are used in a wide variety of settings, including evaluating database performance tuning [15], benchmark development [50], database auditing [54], and compliance validation [32]. Also, many user-centric systems utilize query logs to help users by providing recommendations and personalizing the user experience [22, 38, 51, 89, 96, 104]. As the basic unit of interaction between a database and its users, the sequence of SQL queries that a user issues effectively models the user's behavior. Queries that are similar in structure imply that they might be issued to perform similar duties. Examining a history of the queries serviced by a database can help database administrators with tuning, or help security analysts to assess the possibility and/or extent of a security breach. However, logs from enterprise database systems are far too large to examine manually. As one example, a recent study of queries at a major US bank for a period of 19 hours found nearly 17 million SQL queries and over 60 million stored procedure execution events [54]. Even excluding stored procedures, it is unrealistic to expect any human to manually inspect all 17 million

queries per day.

Let us consider an analyst (call her Jane) faced with the task of analyzing such a query log. Jane might first attempt to identify some interesting query fragments and their aggregate properties. For example, she might count how many times each table is accessed or the frequency with which different classes of join predicates occur. Unfortunately, such fine-grained properties lack the context to clearly communicate how the data is being used, combined, and/or manipulated. To see the complete context, Jane must look at entire queries. Naively, she might look at all *distinct query strings* in the log. Even comparatively small production databases typically log hundreds or thousands of distinct query strings, making direct inspection impractical. Furthermore, it is unclear that distinct query strings are the right level of granularity in the first place. Consider the following example queries:

1. ```
   SELECT name FROM user
   WHERE rank IN ('adm','sup')
   ```

2. ```
   SELECT SUM(balance) FROM accounts
   ```

3. ```
   SELECT name FROM user WHERE rank = 'adm'
       UNION SELECT name FROM user
             WHERE rank = 'sup'
   ```

4. ```
   SELECT SUM(accounts.balance) FROM accounts
       NATURAL JOIN user WHERE user.rank = 'adm'
   ```

Queries 1 and 2 are clearly distinct: Their structures differ, they reference different datasets, and perform different computations. The remaining queries however are less so. Query 3 is logically equivalent to Query 1: Both compute identical results. Conversely, although Query 4 is distinct from Queries 1 and 2, it is conceptually similar to both and shares many structural features with each.

The exact definition of similarity may depend on Jane's exact task, the content of the log, the database schema, database records, and numerous other details, some of which may not be available to Jane immediately when she first begins analyzing the log. It is also likely that some of this information, like the precise contents of the database or even the database schema may not even be available to Jane for reasons of privacy or security. As a result, this type of log analysis can quickly become a tedious, time-consuming process [36]. In this chapter, we lay the groundwork for a more automated approach to query log exploration based on hierarchical clustering. Given a hierarchical clustering of the query log, Jane can manually adjust how aggressively the log is summarized. She can select an appropriate level of granularity without a *priori* needing to specify exactly what constitutes a similar query.

The primary focus of this chapter is to study the suitability of three existing query distance metrics [4, 6, 63] to be used with hierarchical algorithms for clustering query logs. All of these metrics operate on the query structure and do not rely on the availability of underlying data or schema, thus making them applicable in a wide variety of practical settings. We evaluate the three metrics on two types of data: Human-authored and Machine-generated.

For our evaluation, we use three evaluation data sets: i) a large set of student authored queries released by IIT Bombay [20], ii) a smaller set of student queries gathered at the University at Buffalo, and released as part of this publication, and iii) SQL logs that capture all activities on 11 Android phones for a period of one month [50]. Student-written queries are appealing, as queries are already labeled by their ground-truth clusterings — For each question, the student is attempting to accomplish one specific stated task. Conversely, machine-generated queries on smartphones present a conceptually easier challenge, as they produce more rigid, structured queries. The three similarity metrics are evaluated on these data sets using three standard clustering evaluation statistics: Silhouette Coefficient, Beta CV, and Dunn Index [105].

None of the similarity metrics perform as well as desired, so we create and evaluate a pre-processing step to create more regular, uniform query representations by leveraging query equivalence rules and data partitioning operations. This process significantly improves the quality of all three distance metrics. We also investigate and identify sources of errors in the clustering process. Experimental results show that our *regularization* pre-processing technique consistently improves clustering for different query comparison schemes from the literature.

Concretely, the specific contributions of this chapter are: (1) A survey of existing SQL query similarity metrics, (2) An evaluation of these metrics on multiple query logs, and (3) Applying query standardization techniques to improve query clustering accuracy.

5.2 SQL Similarity Metrics

Analyzing query logs mostly relies on the structure of queries [48], although their motivations are different; some methods prefer using the log as a resource to collect information to build user profiles, and the others utilize structural similarity to perform tasks like query recommendation [22, 38, 104], performance optimization [6], session identification [4] and workload analysis [63]. A summary of these methods is given in Table 5.1.

There are also other possible approaches; like data-centric query comparison [64], and utilizing the *access areas* of user queries by inspecting the data partition the query is interested in [72] from the `WHERE` condition. However, these approaches are out of our scope since we are interested in comparing and improving methods based on structural similarity; we assume that we do not have access to the data or the statistical information about the database.

Agrawal *et al.* [3] aim to rank the tuples returned by the SQL query based on the context. They create a ruleset for contexts and evaluate the result of queries that belongs to the

Paper title	Motivation	Features	Feature Structure	Distance Function	Similarity Ratio
Agrawal *et al.* (2006) [3]	Q. reply importance	Schema, rules	Vector	Cosine similarity	No
Giacometti *et al.* (2009) [38]	Q. recommendation	Difference pairs	Set	Difference query	No
Yang *et al.* (2009) [104]	Q. recommendation	Selection/join, projection	Graph	Jaccard coefficient on the graph edges	No
Stefanidis *et al.* (2009) [96]	Data Recommendation	Inner product of two queries	Vector	-	No
Khoussainova *et al.* (2010) [51]	Q. recommendation	Popularity of each query object	Graph	-	No
Chatzopoulou *et al.* (2011) [22]	Q. recommendation	Syntactic element frequency	Vector	Jaccard coefficient and cosine similarity	No
Aouiche *et al.* (2006) [6]	View selection	Selection/join, group-by	Vector	Hamming distance	Yes
Aligon *et al.* (2014) [4]	Session similarity	Selection/join, projection, group-by	3 Sets	Jaccard coefficient	Yes
Makiyama *et al.* (2016) [63]	Workload analysis	Term frequency of projection, selection/join, from, group-by and order-by	Vector	Cosine similarity	Yes

Table 5.1: SQL query similarity literature review

context according to the ruleset. They capture context and query as feature vectors and capture similarity through cosine distance between the vectors.

Chatzopoulou *et al.* [22] aim to assist non-expert users of scientific databases by tracking their querying behavior and generating personalized query recommendations. They deconstruct an SQL query into a bag of *fragments*. Each distinct fragment is a feature, with a weight assigned to it indicating its importance. Each feature has two types of importance: (1) within the query and (2) for the overall workload. Similarity is defined upon

common vector-based measures such as cosine similarity. A summarization/user profile for this approach is just a sum over all single query feature vectors that belong to their workload.

Yang *et al.* [104], on the other hand, build a graph following the query log by connecting associations of table attributes from the input and output of queries which are then used to compute the likelihood of an attribute appearing in a query with a similarity function like Jaccard coefficient. Their aim is again to assist users in writing SQL queries by analyzing query logs. Giacometti *et al.* [38], similarly, aim to make recommendations on the discoveries made in the previous sessions for users to spend less time on investigating similar information. They introduce *difference pairs* in order to measure the relevance of the previous discoveries. Difference pairs are essentially the result columns that is not included in the other return results; hence the method depends on having access to the data. Stefanidis *et al.* [96] takes a different approach, and instead of recommending candidate queries, they recommend tuples that may be of interest to the user. By doing so, the users may decide to change the selection criteria of their queries in order to include these results.

Sapia [89] creates a model that learns query templates to prefetch data in OLAP systems based on the user's past activity. SnipSuggest [51], on the other hand, is a context-aware SQL-autocomplete system that helps database users to write SQL queries by suggesting SQL snippets. In particular, it assigns a probability score to each subtree of a query based on the subtree's frequency in a query log. These probabilities are used to discover the most likely subtree that a user is attempting to construct, at interactive speeds.

Although these methods [3, 22, 38, 51, 96, 104] utilize query similarity one way or other to achieve their purpose, they don't directly offer a way to compare query similarity. We aim to summarize the log and the most practical way to describe a query log is to group similar queries together so that we can provide summaries of these groups to the users. For this purpose, we need to be able to measure pairwise similarity between each query, hence

we need a metric that can do so. As shown in Table 5.1, this condition is only satisfied by [4, 6, 63].

Aouiche *et al.* [6] is the first work we encountered that proposes a pairwise similarity metric between two SQL queries although it is not the aim of their work. They aim to optimize view selection in warehouses by the queries posed to the system. They consider the *selection*, *joins* and *group-by* items in the query to create vectors and use Hamming Distance to measure how similar two queries are. While creating the vector, it doesn't matter if an item appears more than once or where the item is. They cluster similar queries that creates a workload on the system and base their view creation strategy in the system on the clustering result.

Aligon *et al.* [4] study various approaches to defining a similarity function to compare OLAP sessions. They focus on comparing session similarity while also performing a survey on query similarity metrics. They identify *selection* and *join* items as the most relevant components in a query followed by the *group by* set. Inspired by the findings, they propose their own query similarity metric which considers *projection*, *group-by*, *selection-join* items for queries issued on OLAP datacubes. OLAP datacubes are multidimensional models, and they have hierarchy levels for the same attributes. Aligon *et al.* [4] measure the distance between the attributes on different hierarchy levels, and compute the set similarity for *projection*, *group-by*, and *selection-join* sets individually when comparing two queries. In our experiments, since we do not consider the hierarchy levels in an OLAP system but focus on databases, we consider all queries are on the same level in the schema to adjust the formulas presented in the paper. Namely, we compute set similarity of *projection*, *group-by*, *selection-join* sets of two queries with Jaccard coefficient. Also, Aligon *et al.* [4] provide the flexibility to adjust weights of the three feature sets based on the domain needs. We explore how the clustering quality is affected with various weightings in Section 5.5.5.

Makiyama *et al.* [63] approach query log analysis with the goal of analyzing a system's

workload, and they provide a set of experiments on Sloan Digital Sky Survey (SDSS) dataset. They extract the terms in *selection*, *joins*, *projection*, *from*, *group-by* and *order-by* items separately and record their appearance frequency. They create a feature vector using the frequency of these terms which they use to calculate the pairwise similarity of queries with cosine similarity. Instead of clustering, they perform the workload analysis with Self-Organizing Maps (SOM).

To further illustrate how the three structural metrics [4, 6, 63] work, we show the feature representations for the following query for each method in Table 5.2.

```
SELECT u.username, u.yearenrolled
FROM user u, accounts a
WHERE u.id = a.userid
  AND a.balance > 1000
  AND u.id > 20050001
GROUP BY u.yearenrolled
ORDER BY u.yearenrolled
```

In the next section, we construct a generalized feature engineering scheme for query comparison methods to improve the clustering quality. Our work evaluates the performance of the three methods [4, 6, 63] that directly describe a pairwise similarity metric in Section 5.4 due to the lack of performance evaluation for the query similarity metrics in the given studies. We also show that our feature engineering scheme improves the clustering quality with both statistical and empirical methods.

Table 5.2: Representation of three similarity metrics

Paper title	Extracted Feature Vector
Aouiche *et al.* (2006) [6]	{'u.id', 'a.userid', 'a.balance', 'u.yearenrolled'}
Aligon *et al.* (2014) [4]	{'u.username', 'u.yearenrolled'} {'u.id', 'a.userid', 'a.balance'} {'u.yearenrolled'}
Makiyama *et al.* (2016) [63]	{'SELECT_u.username' →1, 'SELECT_u.yearenrolled' →1, 'FROM_user →1', 'FROM_accounts' →1, 'WHERE_u.id' →2, 'WHERE_a.userid' →1, 'WHERE_a.balance' →1, 'GROUPBY_u.yearenrolled' →1, 'ORDERBY_u.yearenrolled' →1}

5.3 Feature Engineering

The grammar of SQL is declarative. By design, users can write queries in the way they feel most comfortable, letting well-established equivalence rules dictate a final evaluation strategy. As a result, many syntactically distinct queries may still be semantically equivalent. Recall example queries 1 and 3, paraphrased here:

1. `SELECT name FROM user`
   `WHERE rank = 'a' OR rank='s'`

3. `SELECT name FROM user WHERE rank = 'a'`
   `UNION SELECT name FROM user WHERE rank = 's'`

Though semantically distinct, these queries produce identical results for any input. Unfortunately similarity of results is not practical to implement: General query equivalence is NP-complete [19] for SQL92 and earlier, while SQL99 and later versions of SQL are turing-complete, due to the introduction of recursive queries.

However, we can still significantly improve clustering quality by standardizing certain SQL features into a more regular form with techniques such as canonicalizing names and

aliases, removing syntactic sugaring, and standardizing nested query predicates. This process of *regularization* aims to produce a new query that is more likely to be *structurally* similar to other *semantically* similar queries. Because the output is an ordinary SQL query, regularization may be used with any similarity metric. These process is similarly used in [21, 88], where Chandra *et al.* [21] generate mutations of SQL queries to catch diversions from a baseline query, and Sapia [88] creates OLAP query prototypes based on selected features and models user profiles.

Although the techniques we utilize for regularization are widely used in other settings, to the best of our knowledge, we introduce their usage to improve clustering quality. We also test all the techniques we use individually to find their impact on the regularization's overall effect. Our experiments in Section 5.5.2 show consistent improvements for all metrics evaluated in practical real world settings. In this section, we describe the transformations that we apply to regularize queries and the conditions under which they may be applied.

5.3.1 Regularization Rules

Canonicalize Names and Aliases. As we will show in our experiments in Section 5.5, table and attribute aliases are a significant source of error in matching. Consider the following two queries:

5. `SELECT name FROM user`

6. ```
   SELECT id
   FROM (SELECT name AS id FROM user) AS t
   ```

Although these queries are functionally identical, variable names are aliased in different ways. This is especially damaging for the three structural heuristics that we evaluate, each of which assumes that variable names follow a globally consistent pattern. Our first
   ```

Before	After
$b\ \{>, \geq\}\ a$	$a\ \{<, \leq\}\ b$
x **BETWEEN** (a,b)	$a \leq x$ **AND** $x \leq b$
x **IN** (a,b,...)	x=a **OR** x=b **OR** ...
isnull(x,y)	**CASE WHEN** x is null **THEN** y **END**

Table 5.3: Syntactic Desugaring

regularization step attempts to create a canonical naming scheme for both attributes and tables which is similar to one used in [21].

Syntax Desugaring. We remove SQL's redundant syntactic sugar following basic pattern-replacements as shown in Table 5.3.

EXISTS Standardization. Although SQL admits four classes of nested query predicates: (`EXISTS`, `IN`, `ANY`, and `ALL`), the `EXISTS` predicate is general enough to capture the semantics of the remaining operators [21]. Queries using the others are rewritten:

`x IN (SELECT y ...)` **becomes**

```
EXISTS (SELECT * ...WHERE x = y)
```

`x < ANY (SELECT y ...)` **becomes**

```
EXISTS (SELECT * ...WHERE x < y)
```

`x < ALL (SELECT y ...)` **becomes**

```
NOT EXISTS (SELECT * ...WHERE x ≥ y)
```

DNF Normalization. We normalize all boolean-valued expressions by converting them to disjunctive normal form (DNF). The choice of DNF is motivated by the ubiquity of conjunctive queries in most database applications, as well as by the natural correspondence between disjunctions and unions that we exploit below.

Commutative Operator Ordering. We standardize the order of expressions involving commutative and associative operators (e.g., $\wedge$, $\vee$, $+$, and $\times$) by defining a canonical order of all operands and traversing the expression tree bottom-up to ensure consistent order of all operands.

Flatten FROM-Nesting. We merge nested sub-queries in a `FROM` clause with its parent query as described in [21].

Nested Query De-correlation. A common database optimization called nested-query de-correlation [92] converts some `EXISTS` predicates into joins for more efficient evaluation. Note that this rewrite does not guarantee query result equivalence under *bag semantics* due to duplicated rows in the result. Hence we require that the parent query is either a `SELECT DISTINCT` or a duplicate-insensitive aggregate [41] (e.g. $max\{1,1\} = max\{1\}$, but $sum\{1,1\} \neq sum\{1\}$). If the `EXISTS` predicate is in a purely conjunctive `WHERE` clause, the de-correlation process simply moves the query nested in the `EXISTS` into the `FROM` clause of its parent query. The (formerly) nested query's `WHERE` clause can be then merged into the parent's `WHERE` clause. Specifically, if the input query is of the form:

```
SELECT ... FROM R WHERE
           EXISTS (SELECT ... FROM S WHERE q)
```

then the output query will have the form:

```
SELECT ... FROM R, (SELECT ... FROM S) WHERE q
```

To de-correlate a `NOT EXISTS` predicate, we use the set-difference operator `EXCEPT`. If the input is of the form:

```
SELECT DISTINCT... FROM R WHERE
       NOT EXISTS (SELECT ... FROM S WHERE q)
```

then the output will be of the form

```
(SELECT DISTINCT... FROM R) EXCEPT
  (SELECT DISTINCT... FROM R, WHERE
           EXISTS (SELECT ... FROM S WHERE q))
```

OR-UNION Transform. We use a regularization transformation that exploits the relationship between OR and UNION. This rewrite does not guarantee query result equivalence, also due to potentially duplicated rows in query result. Recall the equivalence between

logical OR and UNION mentioned in our first example. Naively, we might convert the DNF-form predicates into UNION queries:

`SELECT ... WHERE q OR p OR ...` **becomes**

```
SELECT ... WHERE q UNION SELECT ... WHERE p UNION ...
```

However, duplicates caused by the possible correlation between clauses in DNF will break the equivalence of this rewrite. Consider the following query:

```
SELECT Score FROM Exam WHERE Score>60 OR Pass=1
```

Students who pass the exam overlap with those whose score greater than 60. Thus the rewritten query would not be exactly equivalent, as it may include duplicate rows. As a result, we require the query to satisfy the same condition mentioned in previous rule *nested query de-correlation*.

Union Pull-Out. Since the prior transformation may introduce UNION operator in nested subqueries, we push selection predicates down into the union as well.

5.4 Quality Metrics

In this section, we introduce the quality measures and workloads to evaluate three query similarity metrics and the feature engineering scheme. Our goal is to evaluate how well a query similarity metric captures the task behind a query with and without regularization. We use two types of real-world query workloads: human- and machine-generated. We expect the problem of query similarity to be harder on human-generated workloads, as queries generated by machines are more likely to follow a strict, rigid structural pattern.

As a source of human-generated queries, we use two different sets of student answers to database course assignments. Many database courses include homework or exam questions where students are asked to translate prose into a precise SQL query. This provides us with a ground-truth source of queries with different structures that should be similar. As

machine-generated queries, we use PocketData [50] a log of 33 million queries issued by smartphone apps running on 11 phones in the wild over the course of a month.

In subsection 5.4.1, we outline the datasets used. Then, in subsection 5.4.2, we outline the experimental methodology used to evaluate distance metrics, and show a set of measures for quantitatively assessing how effective a query similarity metric is at clustering queries with similar tasks.

5.4.1 Workloads

We use three specific query sets: Student assignments gathered by IIT Bombay [20], student exams gathered at our department (denoted as UB dataset in the experiments) and released as part of this chapter[1], and SQL query logs of the Google+ app extracted from PocketData dataset [50].

The first dataset [20] consists of student answers to SQL questions given in IIT Bombay's undergraduate databases course. The dataset consists of student answers to 14 separate query-writing tasks, given as part of 3 separate homework assignments. The query writing tasks have varying degrees of difficulty. Answers are not linked to anonymous student identifiers and there is no grade information. The IIT Bombay dataset is exclusively answers to homework assignments, so we expect generally high-quality answers due to the lack of time pressure and availability of resources for validating query correctness.

The second dataset consists of student answers to SQL questions given as part of our department's graduate database course. The dataset consists of student answers to 2 separate query-writing tasks, each given as part of midterm exams in 2014 and 2015 respectively. SQL queries were transcribed from hand-written exam answers, anonymized for IRB compliance and labeled with the grade the answer was given. We expect quality to vary, as

[1] `http://odin.cse.buffalo.edu/public_data/2016-UB-Exam-Queries.zip`

exams are closed-book and students have limited time. Since 50% of the grade is the failing criterion, we assume that answers conform with the task of the question if the grade is over 50%.

The third dataset consists of SQL logs that capture all database activities of 11 Android phones for a period of one month. We selected Google+ application for our study since it is one of the few applications where all users created a workload. SQL queries collected were anonymized and some of the identified query constraints were deleted for IRB compliance [50].

A summary of all datasets is given in Tables 5.4, 5.5, and 5.6. The prose questions asked for IIT Bombay and UB Exam datatsets can be found in Table 5.7 and 5.8. Not all student responses are legitimate SQL, and so we ignore queries that cannot be successfully parsed by our open-source SQL parser[2]. We also released the source code we used in the experiments[3].

In the first two datasets, the query-writing task is specific. We can expect that student answers to a single question are written with the same task. Thus, we would expect a good distance metric to rate answers to the same question as close and answers to different questions as distant. Similarly, using the distance metric for clustering, we would expect to see each query cluster to uniformly include answers to the same question.

In the third dataset, PocketData-Google+, the queries are generated by the Google+ application. Since some of the constants are replaced with standard placeholders for IRB compliance, the number of distinct queries drops significantly. Since there is no information about what kind of a task a query is trying to perform, we inspected and manually labeled each distinct query string. Queries were labeled with one of 8 different categories: Account, Activity, Analytics, Contacts, Feed, Housekeeping, Media and Photo.

[2] `https://github.com/UBOdin/jsqlparser`
[3] `https://github.com/UBOdin/EttuBench`

Question	Total	Parsable	Distinct query strings
1	55	54	4
2	57	57	10
3	71	71	66
4	78	78	51
5	72	72	67
6	61	61	11
7	77	66	61
8	79	73	64
9	80	77	70
10	74	74	52
11	69	69	31
12	70	60	22
13	72	70	68
14	67	52	52

Table 5.4: Summary of IIT Bombay dataset

Year	2014	2015
Total queries	117	60
Syntactically correct queries	110	51
Distinct query strings	110	51
Query with score $> 50\%$	62	40

Table 5.5: Summary of UB Exam dataset

	Pocket Dataset	**Google+**
All queries	45,090,798	2,340,625
SELECT queries	33,470,310	1,352,202
Distinct query strings	34,977	135

Table 5.6: Summary of PocketData dataset and Google+

ID	Question
1	Find course_id and title of all the courses
2	Find course_id and title of all the courses offered by “Comp. Sci." department.
3	Find course_id, title and instructor ID for all the courses offered in Spring 2010
4	Find id and name of all the students who have taken the course “CS-101"
5	Find which all departments are offering courses in Spring 2010
6	Find the course ID and titles of all courses that have more than 3 credits
7	Find, for each course, the number of distinct students who have taken the course; in case the course has not been taken by any student, the value should be 0
8	Find id and title of all the courses offered in Spring 2010, which have no pre-requisite
9	Find the ID and names of all students who have (in any year/semester) taken two courses
10	Find the departments (without duplicates) of courses that have the maximum credits
11	Show a list of all instructors (ID and name) along with the course_id of courses they have taught. If they have not taught any course show the ID and name with null value for course_id
12	Find IDs and names all students whose name contains the substring “sr" ignoring case. (Hint Oracle supports the functions lower and upper)
13	Using a combination of outer join and the is null predicate but WITHOUT USING "except/minus" and "not in" find IDs and names of all students who have not enrolled in any course in Spring 2010
14	A course is included in your CPI calculation if you passed it, or you have failed it, and have not subsequently passed it (or in other words, a failed course is removed from CPI calculation if you have subsequently passed it). Write an SQL query that shows all tuples of the relation other than those eliminated by the above rule, and also eliminating tuples with a null value for grade

Table 5.7: Questions given IIT Bombay Dataset [20]

Year	Question
2014	How many distinct species of bird have ever been seen by the observer who saw the most birds on December 15, 2013?
2015	You are hired by a local birdwatching organization, who’s database uses the Birdwatcher Schema on page 2. You are asked to design a leader board for each species of Bird. The leader board ranks Observers by the number of Sightings for Birds of the given species. Write a query that computes the set of names of all Observers who are highest ranked on at least one leader board. Assume that there is no tied rankings.

Table 5.8: UB Exam dataset questions

5.4.2 Clustering validation measures

In addition to workload datasets, we define a set of measures to be used for evaluating queries. Given a set of queries labeled with tasks and an inter-query similarity metric, we want to understand how well the metric can (1) put queries that perform the same task close together even if they are written differently, and (2) differentiate queries that are labeled with different tasks.

We evaluate each metric according to how well it aligns with the ground-truth cluster labels. Rather than evaluating the clustering output itself, we evaluate an intermediate step: the pairwise distance matrix for the set of queries in a given workload. With this matrix and a labeled dataset, we can use various clustering validation measures to understand how effectively a similarity metric characterizes the partition of a set of queries. Specifically, we will use three clustering validation measures [105, Chapter 17]: Average Silhouette Coefficient, BetaCV and Dunn Index.

Silhouette coefficient. For every data point in the dataset, its silhouette coefficent is a measure of how similar it is to its own cluster in comparison to other clusters. In particular, the silhouette coefficient for a data point i is measured as $\frac{b(i)-a(i)}{max(a(i),b(i))}$ where $a(i)$ is the average distance from i to all other data points in the same cluster and $b(i)$ is the average distance from i to all other data points in the closest neighboring cluster. The range of silhouette coefficient is from -1 to 1. We denote $s(i)$ to represent silhouette coefficient of data point i. $s(i)$ is close to 1 when $s(i)$ is close to other data points from the same cluster more than data points from different clusters, which represents a good match. On the other hand, $s(i)$ which is close to -1 represents that the data point i stayed in the wrong cluster, as it is closer to data points in different clusters than its own. Since the silhouette coefficient represents a measure of degree of goodness for each data point, to validate the effectiveness of the distance metric given a query partition, we use the average silhouette

coefficient of all data points (all queries) in the dataset.

BetaCV measure. The BetaCV measure is the ratio of the total mean of intra-cluster distance to the total mean of inter-cluster distance. The smaller the value of BetaCV, the better the similarity metric characterizes the cluster partition of queries on average.

Dunn Index. The Dunn Index is defined as the ratio between minimum distance between query pairs from different clusters and the maximum distance between query pairs from the same cluster. In other words, this is the ratio between closest pairs of points from different clusters over the largest diameter among all clusters. Higher values of the Dunn Index indicate better the worst-case performance of the clustering metric.

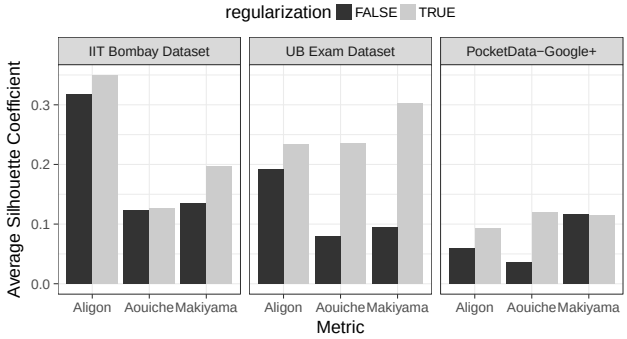
5.5 Experiments

In this section, we perform experiments to evaluate the performance of three similarity metrics previously discussed in Section 5.2: Makiyama's similarity [63], Aligon's similarity [4] and Aouiche's similarity [6]. We implemented each of these similarity metrics in Java and evaluated them using the three clustering validation measures discussed in subsection 5.4.2. In particular, we evaluate these three similarity metrics on their ability to capture the tasks performed by SQL queries. In addition, we also evaluate the effectiveness of the feature engineering step introduced in Section 5.3 and understand how query similarity can be improved by applying this step on the SQL query. We also look closer at feature engineering by breaking it down to different modules and analyze the effect of each module on capturing the tasks performed by queries.

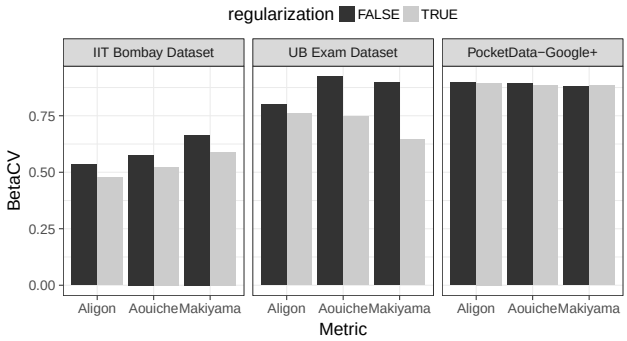
5.5.1 Evaluation on SQL similarity metrics

In the first experiment, we evaluate three similarity metrics mentioned in Section 5.2. The aim of the experiment is to evaluate which similarity metric can best capture the task per-

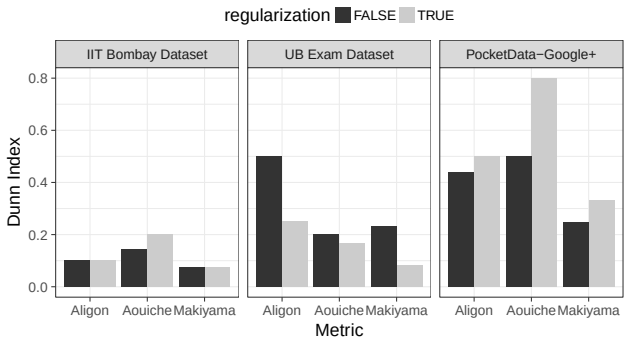
formed by each query.



(a) Average Silhouette Coefficient
(*larger* values are better)



(b) BetaCV
(*smaller* values are better)

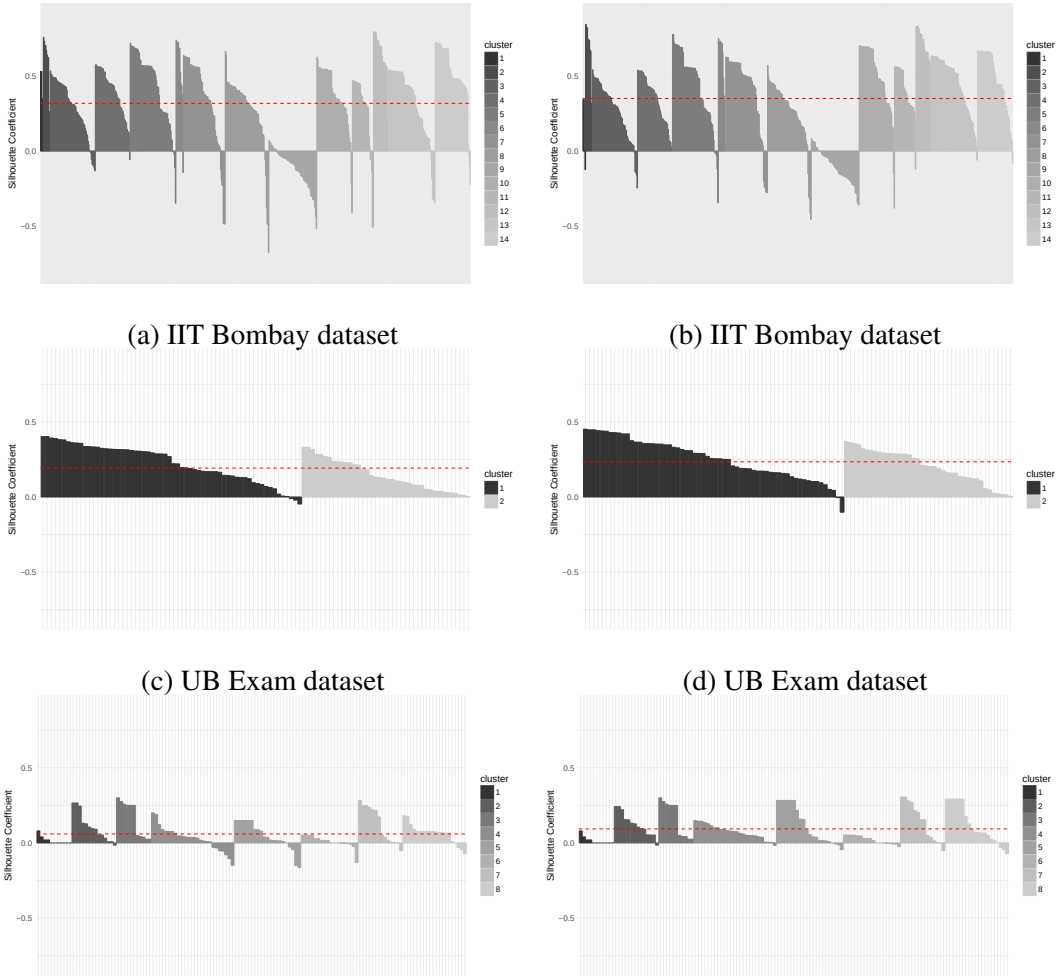


(c) Dunn Index
(*larger* values are better)

Figure 5.1: Clustering validation measures for each metric with and without regularization step

The black columns in Figure 5.1 show a comparison of three similarity metrics using each of the three quality measures (Average Silhouette Coefficient, BetaCV and Dunn Index). As can be seen in Figure 5.1, Aligon seems to work the best for both IIT Bom-

bay and UB Exam dataset while achieving second-best for PocketData-Google+ dataset under the Average Silhouette Coefficient measure. When considering BetaCV measure, Aligon also attains the best result for both IIT Bombay and UB Exam dataset while having comparable result for PocketData-Google+ dataset. Aligon also performs well on the Dunn Index, coming in first on UB Exam dataset, and second-best for IIT Bombay and PocketData-Google+ dataset. Especially given that the Dunn Index measures only worst-case performance, Aligon's metric seems to be ideal for our workloads. This shows that even a fairly simple approach can capture task similarity well.



(a) IIT Bombay dataset (b) IIT Bombay dataset

(c) UB Exam dataset (d) UB Exam dataset

(e) PocketData-Google+ dataset (f) PocketData-Google+ dataset

Figure 5.2: Distribution of silhouette coefficients when using Aligon's similarity without regularization (a,c,e), and when regularization is applied (b,d,f)

For a closer look of Aligon's similarity metric, Figure 5.2(a,c,e) shows the distribution of Silhouette coefficients for each query and their respective tasks. Recall that the silhouette coefficient below 0 effectively indicates a query closer to another cluster than its own, or a query that would be mis-classified. The further below zero, the greater the error. For the UB Exam dataset (Figure 5.2c), the majority of queries would have been successfully classified, and only a small fraction exhibit minor errors. For the PocketData-Google+ dataset (Figure 5.2e), there are some erroneous queries in cluster 4, 5 and 6 while cluster 1, 2, 3, 7 and 8 have very few errors. For the Bombay dataset (Figure 5.2a), the distribution of errors varies. Cluster 1, 2, 4, 6, 12 and 14 exhibit virtually no error, while cluster 7, 8, and 9 exhibit particularly egregious errors.

5.5.2 Evaluation of feature engineering

We next evaluate the effectiveness of regularization by applying it to each of the three metrics described in Section 5.2. We use our quality evaluation scheme to compare the quality of each measure both with and without feature engineering.

Figure 5.1 shows the values of three validation measures for each of the three similarity metrics, both with and without regularization. As shown in Figure 5.1, regularization significantly improves the Average Silhouette Coefficient and BetaCV measures for all similarity metrics except for the case of Makiyama similarity metric with PocketData-Google+ dataset. The Dunn index is relatively unchanged or little improved for the IIT Bombay and PocketData-Google+ dataset and shows slight signs of worsening with regularization on the UB Exam dataset. To understand the reason of worse Dunn Index, we compare Figure 5.2c (original) with Figure 5.2d (with regularization). The Silhouette Coefficient for answers that are originally positive in each question are considerably increased, and for answers that are originally negative (regarded erroneous) are even more decreased as a result

of regularization, since it reduces the query structure diversity which leads to separating queries better. In other words, for erroneous answers with negative Silhouette Coefficients, distance metrics like Aligon distinguish them further apart from answers with positive Silhouette Coefficients after regularization. Since erroneous answers are treated as the 'worst cases' for each question, the Dunn Index which measures worst case performance naturally gets worse.

Per-Query Similarity

Figure 5.2(b,d,f) shows the distributions of silhouette coefficients for the Aligon similarity metric after regularization is applied. For IIT Bombay dataset, comparing against Figure 5.2a there is a slight improvement at the tail end of clusters 9, 11, 12, 13 and 14 — several of the negative coefficients have been removed. Furthermore, positive matches have been improved, particularly for cluster 7, 9, 10, 12 and 13. Finally, there has been a significant reduction in the degree of error in cluster 10. Cluster 10 is a particularly egregious case of aliasing, as the correct answer involves two self-joins in the same query. As a result, aliasing is a fundamental part of the correct query answer, and our rewrites could not reliably create a uniform set of alias names. In the UB Exam and PocketData-Google+ datasets, the improvement provided by regularization can be seen for queries with both positive and with negative values of $s(i)$.

5.5.3 Case Study

As part of our analysis, we attempted to provide empirical explanations for query errors, in particular for queries where $s(i) < 0$ for all three similarity metrics. Namely, we looked into the queries that are too far apart from the clusters they belong, and we categorized the reasons for misclassification based on these queries. We then investigated how the

regularization process particularly affect these queries.

Almost all of these egregiously misclassified queries appear in the IIT Bombay dataset, the distribution of which is summarized in Table 5.9. The PocketData-Google+ dataset includes no egregiously misclassified queries, while the UB Exam dataset includes only one such query (which we tagged as a case of **Contextual equivalence**). We tagged each egregiously misclassified query with an explanation that justifies why the query has a low s(i). Tags were drawn from the following list:

Ground-truth error. A student's response to the question may have been legitimately incorrect. This is a query that is correctly classified as an outlier. For example:

```
SELECT *
FROM (SELECT id, name, time_slot_id
 FROM (SELECT *
  FROM (SELECT *
   FROM student
    NATURAL JOIN takes) b1) a, section
   WHERE a.course_id = section.course_id) a1
```

This query was attempting to complete the task *"Find the ID and names of all students who have (in any year/semester) taken two courses in the same timeslot."*

Nested subquery. A student's response is equivalent to a legitimately correct answer but uses nested subqueries such that a heuristic distance metric cannot recognize. For example:

```
SELECT id, name FROM student
WHERE id IN (SELECT DISTINCT s.id
 FROM (SELECT * FROM takes NATURAL JOIN section) s,
  (SELECT * FROM takes NATURAL JOIN section) t
 WHERE s.id = t.id
  AND s.time_slot_id = t.time_slot_id
  AND s.course_id <> t.course_id)
```

Here, the subquery nesting structure is significantly different from other queries for of

the same question.

Aliasing. Aliasing (e.g., `AS` in SQL) breaks a distance metric that relies on attribute and relation names. For example:

```
SELECT DISTINCT student.id, student.name
FROM student, takes, section AS a, section AS b
WHERE student.id = takes.id
  AND takes.course_id = a.course_id
  AND takes.course_id = b.course_id
  AND a.course_id <> b.course_id
  AND a.time_slot_id = b.time_slot_id
```

The student's use of `a` and `b` make this query hard to distinguish from other queries that may use other names for the attributes.

Insufficient features. Relevant query components are not sufficiently captured as features for a heuristic distance metric to distinguish between answers from sufficiently similar questions.

Too many features. Irrelevant query components create redundant features that artificially increase the distance between the query and cluster center. For example:

```
SELECT DISTINCT student.name, takes.id,
                s1.course_id, s2.course_id
FROM section AS s1, section AS s2, takes, student
WHERE takes.course_id = s1.course_id
  AND s1.course_id <> s2.course_id
  AND s1.time_slot_id = s2.time_slot_id
  AND s1.semester = s2.semester
  AND s1.year = s2.year
  AND takes.sec_id = s1.sec_id
  AND s1.semester = takes.semester
  AND s1.year = takes.year
  AND student.id = takes.id
```

```
  AND s2.time_slot_id = s2.time_slot_id
  AND takes.sec_id = s2.sec_id
  AND s2.semester = takes.semester
  AND s2.year = takes.year
```

Contextual equivalence. Establishing query equivalence to properly clustered queries requires domain-specific knowledge not available to the distance metric (e.g. attribute uniqueness). For example:

```
SELECT student.id, student.name
FROM student
WHERE student.id
  IN (SELECT takes.id
    FROM takes, section
    WHERE takes.course_id = section.course_id
      AND takes.sec_id = section.sec_id
      AND takes.semester = section.semester
      AND takes.year = section.year
    GROUP BY takes.id,
        takes.semester,
        takes.year,
        section.time_slot_id
    HAVING count(*) > 1)
```

Cause	Erroneous queries Without Regularization	Erroneous queries With Regularization
All queries	33 (100%)	27 (100%)
Ground-truth quality	14 (42.4%)	14 (51.8%)
Nested subquery	7 (21.2%)	5 (18.5%)
Aliasing	8 (24.2%)	5 (18.5%)
Insufficient features	2 (6.0%)	1 (3.7%)
Too many features	1 (3.0%)	1 (3.7%)
Contextual equivalence	1 (3.0%)	1 (3.7%)

Table 5.9: Empirical error reasons for IITBombay Dataset

Table 5.9 shows the primary reasons why these queries could not be classified correctly. Note that there may be more than one reason for a query to be placed in a different cluster, but in Table 5.9, we only give the empirically determined primary reason.

Many of the queries with low silhouette coefficients are identified as incorrect answers for the task given. These answers directly affect the ground-truth quality, therefore reduce the average silhouette coefficient. Another reason for erroneous queries with low silhouette coefficients is because of aliasing. Although it is convenient for user to use aliases in the query to refer to a particular item, it is difficult for a machine to approximate the tasks the query authors are trying to accomplish since different query authors have different ways to name particular items in the query. This problem is particularly prevalent in question 9 of the IIT Bombay dataset.

Although the distribution of the error reasons are expected to change, all the tags provided in this section can generically be applied to other query logs given a ground-truth. The regularization method cannot be expected to fix errors originating from misclassifications in ground-truth since they do not actually share any similarities with the cluster.

After the regularization process, the silhouette coefficient under all three similarity metrics for each query is computed again and the result yields an 18% overall reduction in number of erroneous queries ($s(i) < 0$) in the IIT Bombay dataset.

5.5.4 Analysis of regularization by module

In Subsection 5.5.2, we analyzed the overall effect of regularization on query similarity. However, as described in Section 5.3, regularization is composed of many different transformation rules. In this experiment, we group these rules into four separate modules and inspect their impact on the clustering quality. One may observe that Commutative Operator Ordering is guaranteed to provide benefit in structure similarity comparison, hence we in-

clude it in all four modules. In addition, there are dependencies between rules that require them to operate one before another. For example, we should better apply Syntax Desugaring and then DNF Normalization to simplify the boolean expression in WHERE clause before OR-Union Transformation. As another example, Exists Standardization should better be applied on nested sub-queries before we de-correlate them using Nested Query Decorrelation. As a result, we group the rules from Section 5.3 into four modules:

1. *Naming*: **Canonicalize Names and Aliases**

2. *Expression Standardization*: **Syntax Desugaring**,
   Exists Standardization, **DNF Normalization**,
   Nested Query Decorrelation, **OR-Union Transform**

3. *FROM-Nesting*: **Flatten FROM-Nesting**

4. *Union Pullout*: **OR-UNION Pullout**

Commutative Operator Ordering is included in all modules.

Figure 5.3 provides a comparison of each module in regularization. From this figure, one can observe that, since students use different names/aliases for their convenience when constructing queries, the *Naming* module is the most effective one in terms of improving clustering quality for IIT Bombay and UB Exam datasets. On the other hand, for PocketData-Google+ dataset, names are already canonicalized as they are machine-generated. In this case, *Expression Standardization* seems to be the most effective module, especially when using Aligon or Aouiche as similarity metric. In PocketData-Google+ dataset, referred tables and boolean expressions in the queries are both informative in distinguishing between different query categories or clusters. For this reason, Makiyama similarity metric which considers both works well even without regularization while Aligon

and Aouiche can get commensurate performance only after applying *Expression Standardization* module.

Note that in Figure 5.3, *Expression Standardization* makes Average Silhouette Coefficient worse in some cases for IIT Bombay and UB Exam data sets. The performance degradation is majorly due to *feature duplication*. More specifically, consider the example query with *Expression Standardization*.

Example 5. *Syntax Desugaring with OR-UNION transform*

1. `SELECT name FROM usr WHERE`
   `rank IN {'admin','normal'}`
2. `SELECT name FROM usr WHERE`
   `rank = 'admin' OR rank = 'normal'`
3. `SELECT name FROM usr`
   `WHERE rank = 'admin'`
   `UNION`
   `SELECT name FROM usr`
   `WHERE rank = 'normal'`

Query (1) is transformed into (2) by syntax desugaring and then into (3) by OR-UNION Transform. From (1) to (2), feature `WHERE rank` has been replicated; From (2) to (3), features `SELECT name` and `FROM usr` have been duplicated. For expressions of the form: `X IN` $\{x_1, x_2, \ldots, x_n\}$, feature duplication becomes dominant when n grows large. In Figure 5.3, Aligon and Makiyama suffer from feature duplication brought by *Expression Standardization* in some cases while Aouiche does not. Because Aouiche records feature existence instead of occurrence in its vector. Although in some cases such as this, simply replacing feature occurence with existence solves the problem of feature duplication, feature occurence can also be a good indicator for the interests of the query. We believe this problem can be addressed with exploration of feature weighting strategies. Therefore, the problem

of feature duplication will be further explored as a part of feature weighting strategies in our future work.

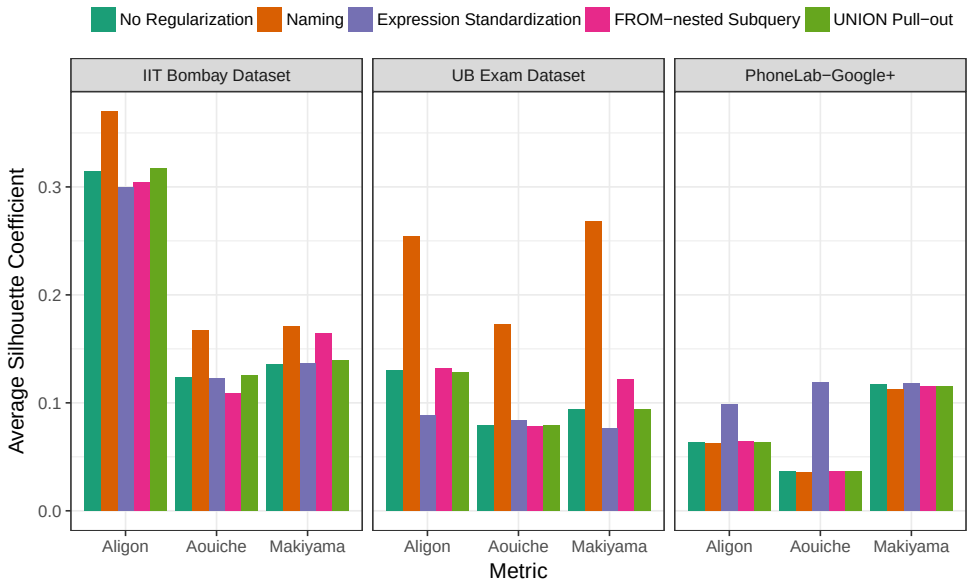


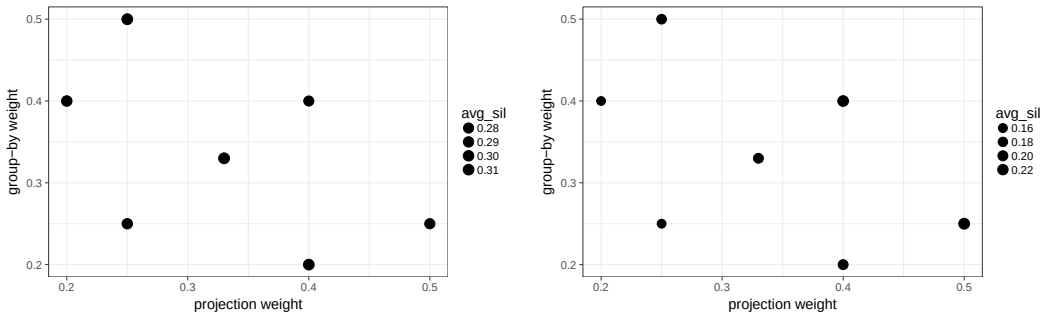
Figure 5.3: Effect of each module in regularization

5.5.5 Analysis of feature set weights

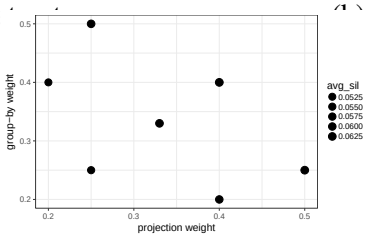
As presented in section 5.2, Aligon's method [4] can be used to compute the similarity between SQL queries. However, Aligon's method requires user to specify the weights of different parts of SQL queries that will be used in calculation of similarity. In this method, a SQL query is divided into three parts including *projection*, *group-by* and *selection-join* (see Table 5.1 and Table 5.2 for details). In this section, we explore the effect of choosing different weight for each query part in Aligon's method.

Let us denote a tuple $(w_{proj}, w_{select-join}, w_{groupby})$ as a set of weights for projection, selection-join and group-by respectively.Totally, we consider seven different combinations of weights including $(0.2, 0.4, 0.4)$, $(0.25, 0.25, 0.5)$, $(0.25, 0.5, 0.25)$, $(0.33, 0.33, 0.33)$, $(0.4, 0.2, 0.4)$, $(0.4, 0.4, 0.2)$ and $(0.5, 0.25, 0.25)$.

Figure 5.4 presents the effect of different sets of weights in three datasets (see Sec-



(a) IIT Bombay Da[illegible] (b) UB Exam Dataset



(c) PocketData-Google+

Figure 5.4: Avergage Silhouette Coefficients with respect to different sets of weights in Aligon method

tion 5.4.1 for descriptions of these datasets). In this experiment, we use average Silhouette coefficient (see Section 5.4.2 for further details) as a measure of quality for each combination of weights. In Aligon method, we have three weights, one for *projection*, *group-by* and *selection-join* respectively, and the total of weights adding up to one. For this reason, we only present *projection* weight and *group-by* weight in Figure 5.4 as *selection-join* weight can be computed by from *projection* and *group-by* weights.

In Figure 5.4, the size (area) of the point indicates the value of average Silhouette coefficient with respect to specific combination of weights. As can be seen in this figure, the values of average Silhouette coefficient change very little when adjusting the weights of projection and group-by. For this reason, in experiments shown in Section 5.5, we use equal weight for each query part, i.e. $(0.33, 0.33, 0.33)$, when applying Aligon's method to different datasets.

5.5.6 Analysis of grading threshold

In this section, we explore different grade cutoff thresholds in UB Exam Dataset and further investigate whether using different values of threshold can significantly affect the experimental result presented in Section 5.5. Table 5.10 gives the summary of number of queries obtained by different levels of threshold. As we can see in the table, the more demanding threshold we have, the fewer valid queries we get from the dataset.

Year	2014	2015
Total number of queries	117	60
Number of queries with score $>$ 20%	110	46
Number of queries with score $>$ 50%	62	40
Number of queries with score $>$ 80%	43	10

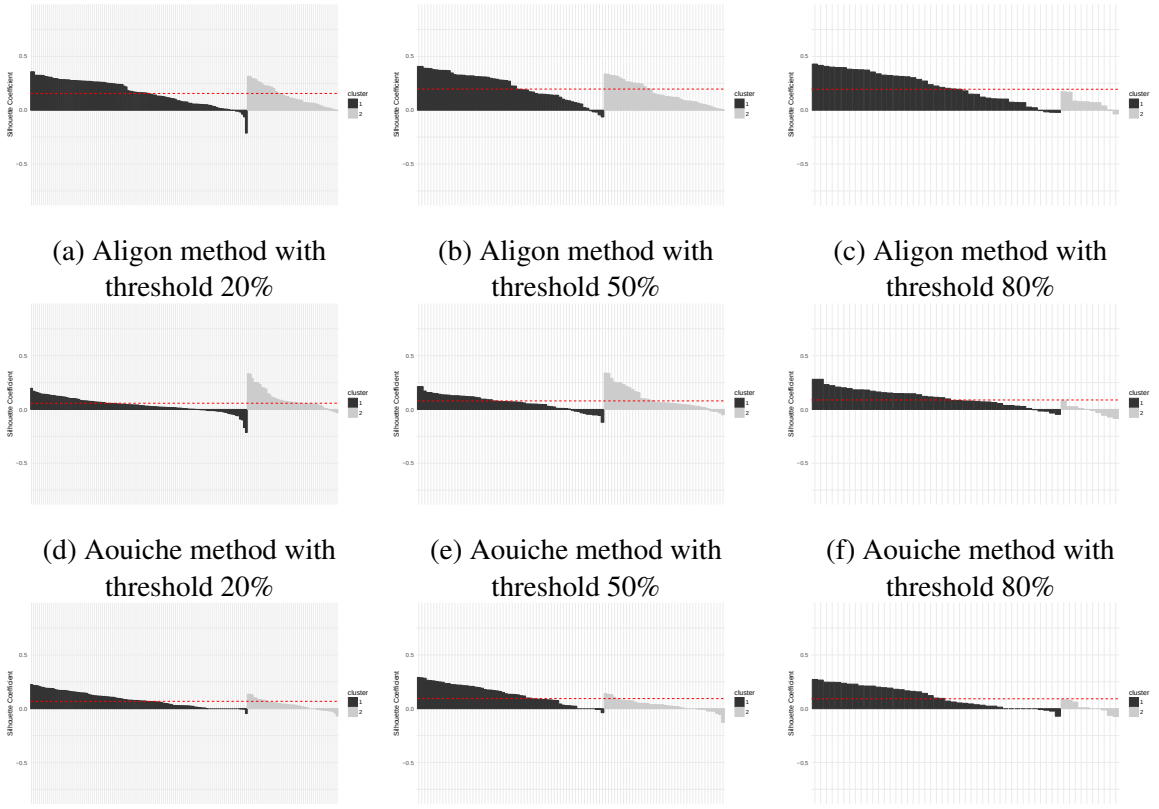
Table 5.10: Number of queries in UB Exam dataset when changing the grading threshold

In Figure 5.5, we show the distribution of Silhouette coefficients of each individual queries in the UB Exam Dataset and see how changing thresholds affects the quality of distance metric. As we can see in this figure, when increasing the grading thresholds, the overall values of Silhoette coeffients also improve. This observation is consistent across all three distance metrics including Aligon, Aouiche and Makiyama. However, as we mentioned earlier, increasing the grading threshold also means fewer queries for analysis. In order to balance between the number of queries and quality of analysis, we choose to use the threshold of 50% in experiments presented in Section 5.5.

5.6 Application Scenarios

In this section, we provide three scenarios where the clustering scheme coupled with the regularization method is applicable.

The first one is, *Jane the DBA* where she takes on the task of improving database per-



(a) Aligon method with threshold 20%

(b) Aligon method with threshold 50%

(c) Aligon method with threshold 80%

(d) Aouiche method with threshold 20%

(e) Aouiche method with threshold 50%

(f) Aouiche method with threshold 80%

(g) Makiyama method with threshold 20%

(h) Makiyama method with threshold 50%

(i) Makiyama method with threshold 80%

Figure 5.5: Silhouette coefficients for UB Exam Dataset when changing grading threshold in three levels - 0.2 (row 1), 0.5 (row 2), 0.8 (row 3)

formance. After performing the straightforward database indexing tasks, she would need to select candidate *views*, which are virtual tables defined by a query. They allow querying just like tables by pre-fetching records from existing tables. Constructing a view for a frequent complex join operation can increase querying performance of the database substantially. To find the ideal views, Jane first clusters similar queries together to see what kinds of queries are more frequent. Making the most frequent complex query types faster by creating views of them could improve database performance substantially [4, 6].

The second one is, *Jane the security auditor* where she suspects that there is a person who leaks classified information from her organization. She can choose to investigate

database access patterns along with other strategies which would involve query clustering [97]. After identifying the query clusters, she can partition the queries by the department or role to get the intuition about which departments and roles *normally* utilize what part of the database. She can detect the *outliers* from that behavior in order to determine the suspects for further investigation.

Lastly, *Jane the researcher* where needs to investigate the properties of the SQL query dataset that she is going to use for her research. One of the new graduate students in her team clusters the queries, and provides her with the clustering assignments of each query. She doubts the quality of the clustering performed, and wonders if the clustering operation could be performed better.

Having a *better* clustering of queries would potentially enhance the quality of her work in all of the examples given above. Also, works cited in this section [4, 6, 97], along with many others can benefit from the framework described in this chapter.

5.7 Discussion

We have reviewed several similarity metrics for clustering queries and focused on three syntax-based methods that offer an end-to-end similarity metric. The advantage of this preference is that, syntax-based methods do not require access to the data in the database or database properties. Considering that only logs are usually transferred between organizations, and requiring access to the data for investigations can cause privacy violations, we preferred focusing on the syntax-based approach.

The survey we performed shows that most of the metrics make use of selection and join operations in the queries and consider them as the most important items for similarity calculation. Group-by aggregate follows them closely while projection items take the third most important item set. There are other possible feature sets that can be used, such as

tables accessed or the abstract syntax tree (AST) of a query, but these feature sets are generally overlooked.

Although Aouiche *et al.* [6] make use of the most important features selection, joins, and group-by items, they don't utilize the number of times an item appears, or after the parsing, they don't consider what kind of feature an item is. This means, it does not matter if a query has `rank` column in group-by, and the other one has `rank` column in selection; they are considered the same. Makiyama *et al.* [63], on the other hand, follow Aligon *et al.* [4] in separating the different features, and improves on it by making use of appearance count of items. However, while trying to make use of every item like `FROM` and `Order-By` predicates, they consider these low priority predicates with same importance as the selection and join predicates.

Makiyama *et al.* [63] use a more complete structure of the query AST, hence when the query is simple like in the PocketData-Google+ dataset, this technique can be slightly better. However, for a complex query with redundant features, mixing features captured from various components of a query without proper feature re-weighting will essentially decrease the weight of features that are more informative. Hence, in student exam datasets, we can observe that Aligon *et al.* [4] is better than the others while in PocketData-Google+ dataset, Makiyama *et al.* [63] is better.

We could further improve these methods by making use of the abstract syntax tree (AST) of a SQL statement. As a declarative language, the AST of a SQL statement acts as a proxy for the task of the query author. This suggests that a comparison of ASTs can be a meaningful metric for query similarity. For instance, we can group a query Q with other queries that have nearly (or completely) the same AST as Q. This structural definition of task has seen substantial use already, particularly in the translation of natural language queries into SQL [61]. For two SQL queries Q_1 and Q_2, one reasonable measure might be to count the number of connected subgraphs of Q_1 that are isomorphic to a subgraph of

Q_2. Subgraph isomorphism is NP-complete, but a computationally tractable simplification of this metric can be found in the Weisfeiler-Lehman (WL) Algorithm [54, 93].

As can be seen in Tables 5.4 and 5.5, as the complexity or difficulty of the question increases, the number of distinct queries also increases, i.e., students find different ways to solve the same problem. Especially, in Table 5.5, no two students answer a question using the same structure. This phenomenon motivates the need for regularization in comparing SQL queries. As the complexity of the query increases, the possible ways to create the query to achieve the same task increase. Figure 5.1 shows that our assumption that regularizing queries will improve overall clustering quality is correct. Our feature engineering scheme improves the overall clustering quality of all three metrics on all three datasets, including both human- and machine-generated queries.

5.8 Summary

The focus of the work of this chapter is to understand and improve similarity metrics for SQL queries relying on query structure to be used to cluster queries. We described a quality evaluation scheme that captures the notion of query task using student answers to query-construction problems and a real-world smartphone query load. We used this scheme to evaluate three existing query similarity metrics. We also constructed a feature engineering technique for standardizing query representations. Through further experiments, we showed that different workloads have different characteristics and no one similarity metric surveyed was always good. The feature engineering steps provided an improvement across the board because they addressed the error reasons we identified.

Chapter 6

Temporal User Behavior Shift

Data leakage, caused by intentional attacks as well as unintentional actions, is one of the most prevalent problems for organizations, including military, intelligence, and business enterprises. One of the crucial reasons why it is very difficult to deal with data leakage is the trust aspect of the legitimate actors (i.e., employees). When an employee misuses the legitimate access rights, gains unauthorized access to a resource, steals someone else's access credentials, or even unintentionally gets their access credentials stolen, they become an *insider threat* [11].

Database access control policies in place are effective to an extent; they can protect sensitive data from being stolen by people who do not have access rights to the data. However, it is almost impossible to create a fine-grained access policy for everyone in a large organization that provides task flexibility for employees, and at the same time, ensures that everybody can access only what they need in order to do their duties. Restrictive access control policies can prevent employees to take on new tasks, while permissive ones can lead to data leakage. Even restrictive policies cannot prevent all types of data leakage; the employee can still access data that they do not need to access to perform a task.

Another way to approach this problem is to monitor database access activity of the

users [14]. This approach allows the security teams to monitor unusual activity, and take precautions against potential threats. The advantage of this approach is to be able to observe all, including permissible, activity. In fact, there are many regulations that require organizations to monitor data access activity such as PCI-DSS, SOX, and HIPAA [69]. Current monitoring mechanisms depend on detecting anomalies in a user's behavior, where the user behavior is captured in different ways [48, 83, 97, 101]. The basic pathway for these systems is to: (1) extract relevant features that reflect the user behavior, (2) cluster similar actions, and (3) find outlier actions, which appear to be a very effective approach. However, such solutions face a crucial shortcoming: they do not consider the *normal drift in the user behavior over time*. Ignoring this key aspect can result in high false alarm rates. For instance, in large companies or government organizations, there are usually different units to perform different functions. Individuals and autonomous systems working for these units perform a variety of tasks; some of the roles assumed in these units can be very repetitive and straightforward while some of them require complicated operations, and they can vary in nature.

Scenario 1. *Let us consider an employee working in the finance department of a company (let's call her Jennifer) who works on billing services, and keeps track of payments. As tax season approaches, she takes on the task to report to the management about the status of the sales, and payments. In order to create the report, Jennifer systematically accesses a variety of types of data from the organization's database while collecting the information required through the organization's ERP system.*

This behavior would result in the ERP system generating queries that access large volumes of data, and different parts of the database. Although these activities are completely legitimate, and expected from Jennifer, anomaly detection systems that are trained with data collected in a different time frame would classify these activities as anomalous, hence

not be able to adapt into new duties and tasks.

As another example, web or smartphone applications can change over time with small modifications, forcing the users to accomplish the same task in a different way [62]. Also, the users can get more proficient with the use of the web or smartphone application in time, or their interests can shift over time. Both of these cases would require the deployed security mechanisms to adapt to the change.

Scenario 2. *Let us consider a photographer (let's call him Jason) using Instagram profile as a portfolio. During the first month of the account opening, he aggressively posts his existing photographs, and builds a portfolio. He then uses the Instagram profile to answer the questions, and communicate with the potential customers. A few weeks later, Instagram introduces a new feature called "Story", which enables users to post short videos and pictures that disappear automatically after 24 hours. Considering taking quality photographs that attract customers takes time and energy, he less frequently posts photos, and he starts to post stories while on the job, to keep the interest of his followers.*

Jason's activity results in Instagram web services to query the database in three different areas of the database: (1) Permanent photo storage, (2) Messaging storage, and (3) Story storage. However, in the first month, his activity constitutes mostly inserting to the permanent photo storage, and increasing read and write access to the messaging storage. After that time period, the queries generated for his activity target mostly inserting to and deleting from the story storage while maintaining a similar workload on messaging, and less frequently generating insert queries to the permanent photo storage. Although both of these behavior shifts are expected, the shifts from normal profile activity can be identified as an outlier if the monitoring system is not designed to adapt to the changes.

A work-around for this problem could be re-training with more recent data when the system starts to return false positives. However, an adaptive system can allow this flex-

ibility, hence reduce false positives, and eliminate the need for retraining. We observe that various experiments reported in the literature do not consider the changes that happen over time, and the synthetic datasets used for the experiments do not reflect any behavior changes.

In this chapter, we bridge this gap by constructing a user behavior model which explicitly models *temporal behavior drift*. We focus on detecting data breaches against relational database systems in an organization to scope down the problem. We argue that this approach is effective in reducing the false positive rates while achieving the same or better performance with regards to true positive rates, compared to the ordinary threshold based approach. In our experiments, we utilize real-world query logs to understand the behavior drift, and validate the effectiveness of our system.

6.0.1 Our Techniques

To identify the behavior drift, and determine if there is a possible data leakage in a user's workload, we defined four basic steps:

Extract SQL query features. We observe and process every SQL query issued to the DBMS. We extract the relevant features of the query considering what part of the database the user is accessing with that particular query. Our work follows the basic principles of the four common SQL query feature extraction methods; Aouiche *et al.* [6], Kamra *et al.* [48], Aligon *et al.* [4], and Makiyama *et al.* [63].

Model user behavior. We construct user profiles by accumulating the extracted features for each user for a given period of time. While doing their daily jobs, people access certain information on the database either through a program, or through a SQL querying interface. It is only expected that there is consistency between queries issued on the database for performing similar duties. The distribution of the features harvested from these queries

constitutes a user profile. The similarity between user profiles, and the difference between distributions are computed with Kullback–Leibler Divergence [58]. We also label each query with its original owner, and use Long Short Term Memory (LSTM) Networks in order to create separate user profiles trained with labeled query vectors.

Identify constant drift. In a large organization, people in different roles usually perform duties that are quite diverse in nature. For example, a database administrator (DBA) can issue a variety of self created queries to the database while a cashier can only issue queries through their cash register that can only create a limited number of preprogrammed queries. Conversely, people who work in the same role in the organization can have different work habits, styles, and priorities. Hence, the expectation of drift in behavior changes for different roles, and for different people.

We take the model function of linear regression for the Kullback–Leibler Divergence scores over time as the expected drift for each profile.

Analyze outliers. Since behavior change is not identical for everyone, it is not viable to expect the detection system to perform well for everyone when run with constant thresholds to determine outliers. Hence, we consider the temporal behavior drift of each user to set an adaptive threshold. When a drift is observed, it is important to identify what the drift is about, and if it is an outlier, or if it is within the acceptable boundaries. We define an anomaly as a value larger than the sum of the expected drift and expected error (*i.e., standard deviation*).

The definition of an anomaly is different for LSTMs – we consider that a part of the user activity is an outlier when the model determines that the incoming query activity belongs to another user.

6.0.2 Contributions and Chapter Outline

In summary, the contributions of this chapter are: (1) Creating a framework to construct normal behavior models of users on a database, (2) Introducing *behavioral drift* in user activity, (3) Comparison of this methodology with LSTMs, (4) Building a threat model for insider and external threats against relational databases, and (5) Providing a defense strategy against the modeled threat. Additionally, to the best of our knowledge, our work is the first study that uses a large real–world SQL query trace to validate the results.

This chapter is organized as follows. Section 6.1 explains the techniques that we use to create user profiles. Section 6.2 presents the experiments performed to show the effectiveness of the system. Finally, we conclude, and discuss our plans to improve our system in Section 6.3.

6.1 Methodology

In this section, we address the threat model described in Chapter 3 with our user behavior modeling methodology, and the anomaly detection strategy.

6.1.1 SQL Query Feature Extraction

Auditing SQL query logs of databases is being used for various purposes from security [97] to database performance optimization [15]. The query logs generally include usernames, timestamps, and other parameters along with the query, hence letting the auditor to collect useful information. However, query interpretation, namely, understanding the goal of the query, is regarded as hard as creating a new query, and even more so for complex queries [36]. Furthermore, complex queries are not uncommon due to the expressive nature of the SQL. The databases are designed and optimized for performance and correctness,

which requires simple relations. Hence, the queries need to be designed more complex with high numbers of table joins as the need increases to access complex information.

There is a line of research that aims to capture user intention through queries since it would contribute to security applications [54], automated personalized query generation [22, 51], and interest mining [96]. To accomplish this, it is essential to identify the required features to be extracted from the SQL queries. As mentioned before, the data stored in the database can also be a good indicator for measuring the similarity of the queries [64]. For instance, consider the following queries:

```
(1) SELECT * FROM taxpayer
    WHERE state LIKE "%california%"
(2) SELECT * FROM taxpayer
```

The first query reads as *list all the taxpayers in California*, and the second query reads as *list all taxpayers*. However, if this query runs on California Tax Service Center, it could be expected that all the taxpayers recorded in the table have "California" in their state column. Thus, SQL queries are open to varying *interpretations*. Consequently, it is crucial to have a SQL query extraction strategy according to why these features are required. For instance, query recommendation [22, 38, 51, 104] requires analysis of feature correlation and dependency, while performance optimization requires discovery of table joins [4].

As discussed before, we observe and process every SQL query issued to the DBMS. We extract the relevant features of the query considering which part of the database the user is accessing with that particular query. Our work follows the basic principles of the four common SQL query feature extraction methods [4, 6, 48, 63].

Aouiche *et al.* [6] aim to optimize view selection in data warehouses by the queries posed to the system. They consider the *selection*, *join* and *group-by* items in the query to create vectors and use Hamming Distance to measure how similar two queries are. While

creating the vector, it doesn't matter if an item appears more than once or where the item is. They cluster similar queries that create a workload on the system and base their view creation strategy in the system on the clustering result.

Kamra *et al.* [48] use database query logs to detect insider attacks. They construct a *quiplet* out of SQL queries, which essentially acts as a vector. The quiplet has 5 different substructures: *command*, a set of tables used in *projection*, projected attributes, a set of tables used in *selection* and *join*, and attributes used in the selection and join. To be able to construct a quiplet when a new query arrives, it is required to possess the database schema structure.

Aligon *et al.* [4] survey on comparing OLAP sessions considering the query similarity, and session similarity. They classify *selection* and *join* attributes as the most relevant component in a query followed by the *group-by* attributes. With the light of the findings, they propose their own SQL query extraction schema which considers *projection*, *group-by*, *selection* and *join* attributes for queries issued on OLAP datacubes.

Makiyama *et al.* [63] perform a set of experiments on Sloan Digital Sky Survey (SDSS) dataset which is a large query log gathered from SDSS systems. There are both human written and machine generated queries. However, there is no classification or ground truth for the queries in the dataset. With the aim of analyzing the workload using query logs, they extract the attributes in *selection*, *join*, *projection*, *from*, *group-by* and *order-by* items separately, and record their appearance frequency.

We approach query feature extraction with the goal of understanding which part of the database the query writer is interested in. We extract the terms in *selection*, *joins*, *projection*, *group-by*, and *order-by* items along with *constants* and *parameters* in the query separately, and record their appearance frequency.

To further illustrate how the three structural metrics [4, 6, 63] work, we show the feature representations for the following query, which reads "*list the username and enrollment year*

in the order of the year of all account owners with a balance higher than $1000 if their ID numbers are higher than 20050001", for each of the methods in Table 6.1.

```
SELECT u.username, u.yearenrolled
FROM user u, accounts a
WHERE u.id = a.userid
  AND a.balance > 1000
  AND u.id > 20050001
GROUP BY u.yearenrolled
ORDER BY u.yearenrolled
```

6.1.2 Normal Behavior

Users access information on the database either through a program, or through a SQL querying interface like MySQL Workbench or a terminal screen, which are then processed in the organization's database servers. Autonomous systems, on the other hand, usually issue the preprogrammed data access requests directly to the organization's database servers.

Building user profiles through clustering, and other machine learning techniques has been studied extensively in the literature before. However, this approaches are not suitable to make the user profiles adapt to the behavior changes, or allow the anomaly detection strategy to consider a possible behavior shift. They usually take a snapshot of the activity at a certain time, and create a model based on the information available at that point of time. Since the query set is clustered with an uncertain number of labels, it is required to compute a pairwise distance matrix between queries to perform a clustering with hierarchical clustering or a similar clustering method. This operation has quadratic complexity [17], and is required to be performed over the whole set of queries. When the model starts to perform worse, the re-training of the model requires the same operation to be repeated.

Table 6.1: Representation of five feature extraction methods

Paper	**Extracted Feature Vector**
Aouiche *et al.* (2006) [6]	{'u.id', 'a.userid', 'a.balance', 'u.yearenrolled'}
Kamra *et al.* (2007) [48]	{'select', <1, 1>, <[0, 1, 1], [0, 0]>, <1, 1>, <[1, 0, 0], [1, 1]>}
Aligon *et al.* (2014) [4]	{'u.username', 'u.yearenrolled'} {'u.id', 'a.userid', 'a.balance'} {'u.yearenrolled'}
Makiyama *et al.* (2016) [63]	{'SELECT_u.username' →1, 'SELECT_u.yearenrolled' →1, 'FROM_user →1', 'FROM_accounts' →1, 'WHERE_u.id' →2, 'WHERE_a.userid' →1, 'WHERE_a.balance' →1, 'GROUPBY_u.yearenrolled' →1, 'ORDERBY_u.yearenrolled' →1}
Our method	{'SELECT_u.username' →1, 'SELECT_u.yearenrolled' →1, 'WHERE_u.id' →2, 'WHERE_a.userid' →1, 'WHERE_a.balance' →1, 'CONSTANT_1000' →1, 'CONSTANT_20050001' →1, 'GROUPBY_u.yearenrolled' →1, 'ORDERBY_u.yearenrolled' →1}

Learning behavior drift. We focus on observing behavior in individual profiles to show the importance of behavior drift. For each user, we define a user profile for a given time-frame T, denoted as the vector $\phi_u^T \in \mathbb{R}^d$, where d is the total number of features extracted using the methodology given in Section 6.1.1. To compute ϕ_u^T, we consider all queries issues by the user u to the database within the timeframe T. A query issued at time t, is a d length vector of counts, and is denoted as q_u^t, where the i^{th} element, $q_u^t[i]$, is equal to the number of times the feature i is observed in the corresponding query.

Note that the feature extraction from a query is an $O(d)$ time complexity operation

where d is a relatively small number, compared to the number of queries.

By combining the feature counts across all queries issued by the user u in a given timeframe, one can compute the entries in the user profile vector, ϕ_u^T, as follows:

$$\phi_u^T[i] = \frac{\sum_{\forall t \in T} q_u^t[i]}{\sum_{j=1}^{d} \sum_{\forall t \in T} q_u^t[j]} \tag{6.1}$$

The *user profiles* are created with the accumulation of these features for a given period of time. Using the appearance frequency of these features, we calculate the appearance probability of each harvested feature. One can also consider the user profile for timeframe T, as a multinomial random variable, which can take one out of d possible values, with probability distribution parameterized by ϕ_u^T.

Given that the features are stored in a map structure, the features of a new query can be simply added to the feature counters which are used to compute the probability of a feature. Hence, this operation has only $O(1)$ time complexity.

Logically, we expect the preprogrammed queries to be more consistent between each other, while handwritten queries to form a more diverse distribution. As we mentioned before, some roles require adaptation to new tasks which would increase the diversity of the features in the user profile. For instance, DBAs and data analysts access a variety of data as required by their jobs. However, an HR intern whose job is only retrieving the applications to the company and reporting them to their superior, diversity is expected to be very low. We define this expected change in behavior with the term *profile drift*.

Comparison of the accumulated user profile, for timespan T_1, with the new incoming behavior observed for timespan T_2, using Kullback–Leibler Divergence [58] gives the *drift score* denoted as follows:

$$d_u^{T_2}(\phi_u^{T_2} || \phi_u^{T_1}) = \sum_i \phi_u^{T_1}(i) log_2 \frac{\phi_u^{T_1}(i)}{\phi_u^{T_2}(i)} \tag{6.2}$$

KL-Divergence (*i.e., relative entropy*) is used for comparing two probability distributions, P and Q; and it ranges between 0 and ∞. $D_{KL}(P||Q)$ essentially represents the information loss when Q distribution is used to approximate P.

Note that when $P(i) \neq 0$ and $Q(i) = 0$, $D_{KL}(P||Q) = \infty$. For example, suppose, we have two distributions P and Q as follows: $P = \{f_0 : 3/10, f_1 : 4/10, f_2 : 2/10, f_3 : 1/10\}$ and $Q = \{f_0 : 3/10, f_1 : 3/10, f_2 : 3/10, f_4 : 1/10\}$. In this case, since f_3 is not a part of Q, the result would be ∞, which means these two distributions are completely different.

Smoothing. To get past this problem, we can apply *smoothing* (i.e., Laplace/additive smoothing), which is essentially adding a small constant $epsilon$ to the distribution, to handle zero values, without significantly impacting the distribution. After we apply smoothing, $D_KL(P||Q)$ becomes 1.38.

The intuition behind using KL-Divergence in our method is to identify the change we experience in the newly coming behavior, compared to the base profile. Similarly, the intuition behind using smoothing is to assume that even if a feature has not been seen in the given dataset, we can still take into account the possibility of its appearance, although very small. Without smoothing, distributions with thousands of matching features except one could be regarded as not related.

Long Short Term Memory (LSTM) Learning. Recurrent Neural Networks (RNN) address the issue of traditional neural networks not being able to consider previous information fed to them. They have loops in them that allow information to persist through time. LSTMs are a special kind of RNNs – they are able to connect long term information that is related to the current state. Under the assumption that user behavior shifts over time, this persistence helps the trained model to observe and consider the change in time.

Similar to the drift model, we create user profiles to train LSTMs. For each user, we define a user profile for a given timeframe T, denoted as the vector $\phi_u^T \in \mathbb{R}^d$, where d is the total number of features extracted using the methodology developed in Section 6.1.1.

To compute ϕ_u^T, we consider all queries issues by the user u to the database within the timeframe T. A query issued at time t, is a d length vector of counts, and is denoted as q_u^t, where the i^{th} element, $q_u^t[i]$, is equal to the number of times the feature i is observed in the corresponding query. However, this time, we label each query with the user ID, and train the LSTM using the series of vectors belonging to every user.

6.1.3 Anomalous Behavior

The user profile evaluates as the new features from the newly coming behavior are imported to the profile. However, before they are added to the profile, they are tested to see if this new activity is an anomalous behavior.

Drift profile model. The drift scores over time, which form a vector denoted as DS, are used to calculate the linear regression coefficients to see the ordinary behavior change for the user. The resulting model function of linear regression of these drift scores for a given period of time yields the *profile drift*, formulated as follows:

$$\hat{DS}_i = \hat{\beta}_0 + \hat{\beta}_1 t_i + \hat{\epsilon}_i \tag{6.3}$$

where $\hat{\beta}_0$ is the y-axis intercept constant, and $\hat{\beta}_1$ is the slope of the profile drift line. $\hat{\epsilon}_i$ is a very small number that represents the noise.

To compute $\hat{\beta}$, given a matrix $C \times N$, the naive least squares computation has overall $O(C^2N)$ complexity, or we can use LU or Cholesky decomposition which takes $O(C^3)$ where C denotes the number of features and N denotes the number of training examples. Since we can usually assume $N > C$, $O(C^2N)$ dominates $O(C^3)$. As a result we can consider that the total asymptotic complexity for linear regression is $O(C^2N)$. However, since we are using the KL-Divergence score of two probability distributions, the number of features scales down to 1, which results in the complexity of this step scaling down to

$O(C^2)$ where C is expected to be a very low number by computational standards. For instance, if we take the profile drift computation interval as one day, we end up with $C = 365$ for a year of data.

Profile drifts occur as the users take on different tasks, as they start to grow different interests, or as they gain experience on the job. Consequently, when this constant change is not addressed properly, utilizing a predefined threshold value can lead to raising too many false positives for the security personnel to inspect when it is set too low, or too many false negatives when it is set too high to avoid false positives. Hence, we define an anomaly as a drift score larger than the sum of the expected profile drift at that specific point, and expected error (*i.e., standard deviation*) as follows:

$$func(\phi_u^{T_2}) = \begin{cases} raisealarm, & \text{if } d_u^{T_2} > \hat{DS}_i + \sigma_{DS} \\ normalbehavior, & \text{otherwise} \end{cases} \tag{6.4}$$

Positive drift in a profile implies that the user is inclined to change their behavior rapidly. Negative drift at any point intuitively suggests that the user started not to get out of their usual pattern as much as before. Issuing no queries at all does not cause any security concerns while increasing the sensitivity to behavior drift when the user starts to issue queries again. When a system uses our model, by using a sliding window strategy, this high sensitivity will be fade away as the behavior drift will converge in time.

LSTM profile model. This deep learning method can determine the label of a given unlabeled query from the label pool – in this case, the user ID. However, it cannot directly say that if a query or query set is an outlier. Hence, we consider a query or query set an outlier if the LSTM determines them belonging to another user than the provided user label in the test data. The long-short term memory provides the system with the capability of considering the query order and time, as well as the preceding queries.

Consequently, a certain set of queries can be determined benign in one part of the query log, and it can be determined as an anomaly in another part.

6.2 Experiments

In this section, we first describe our experimental setup, and the dataset. Then, we summarize our results, and findings from the evaluation of our framework with a real-world SQL query workload.

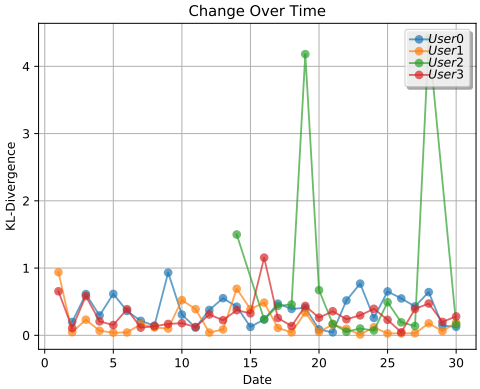
6.2.1 Experimental Setup

In our experiments, the tests were performed on macOS Sierra v10.12.4 on a 2.7 GHz Intel Core i5 with 8GB memory. All of the implementations are performed with Java 1.8 and Python 3.5.

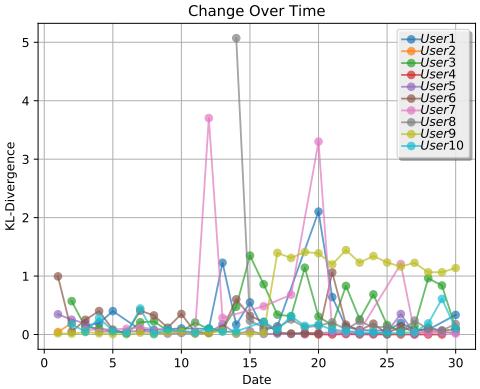
6.2.2 Dataset

The dataset properties we need for this research are threefold: (1) users should use an application to interact with the database, (2) there should be different tasks that the users can perform, and (3) there should be multiple users using multiple applications to be able to investigate if our findings hold for different settings.

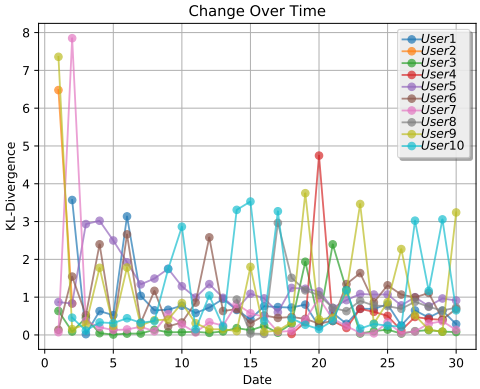
We use smartphone query SQL query logs in our experiments. The experiment dataset consists of SQL logs that capture all database activities of 11 Android phones for a period of one month. SQL queries collected are anonymized, and some of the identified query constraints are deleted for IRB compliance. In this dataset, the queries are generated by the Android applications. There are 45,090,798 queries in total in this dataset. Since some of the constants are replaced with standard placeholders for IRB compliance, the number of



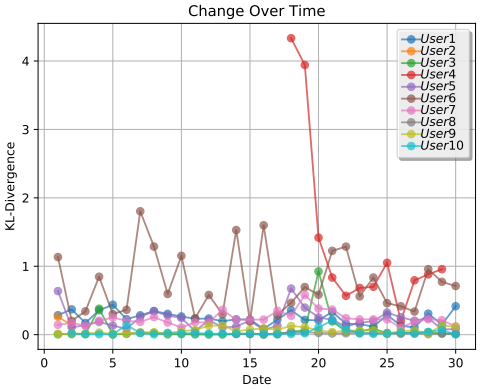
(a) Facebook



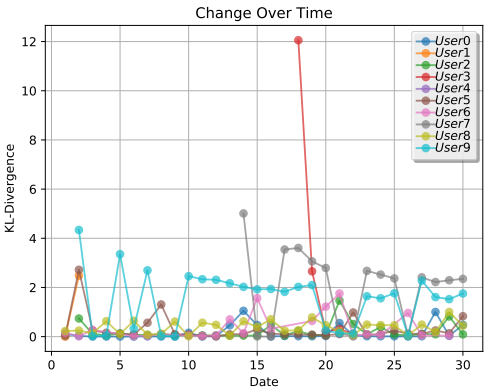
(b) Google+



(c) Hangouts



(d) Google Play Services



(e) Media Storage

Figure 6.1: User behavior change for different Android applications

distinct queries drops significantly. With the help of the operating system, some Android applications can block query monitoring with privacy concerns such as banking applica-

Table 6.2: Dataset

Application	# of queries
Complete Dataset	45,090,798
Facebook	1,272,779
Google+	2,040,793
Hangouts	974,349
Google Play Services	14,813,949
Media Storage	13,592,982

tions. To be able to use all the queries issued by the applications, we selected several of them that do not interfere with the query monitor, and also produced the largest number of queries among all other applications: (1) Facebook, (2) Google+, (3) Hangouts, (4) Google Play Services, and (5) Media Storage.

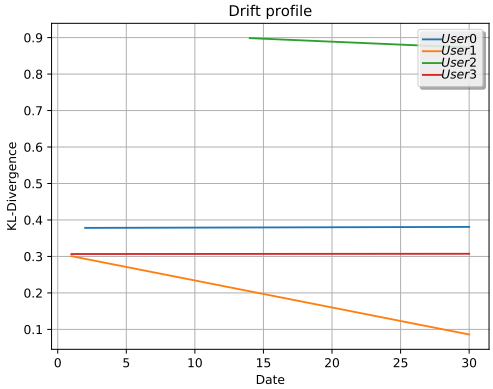
Hence, the smartphone query logs satisfy all of the defined requirements by providing insights to the behavior of 11 users on 5 different Android applications with varying query generation strategies over a period of a month. We can also perform the same experiments on smaller, less used applications. The total query numbers for each application can be seen in Table 6.2.

Not all queries issued by Android applications are legitimate SQL; there can be stored procedure calls, and environment variable checks. The query logs we used for our experiments are available online [1]. We ignore queries that cannot be successfully parsed by our open-source SQL parser [2]. We also released the source code we used in the experiments [3].

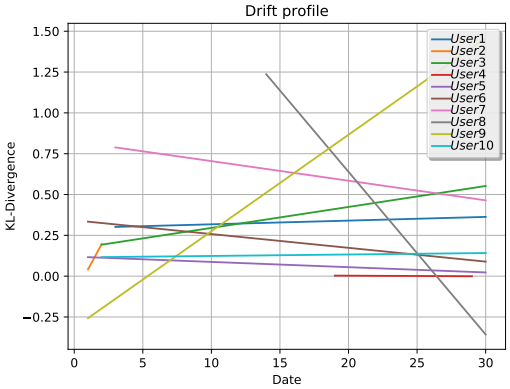
6.2.3 Per-User Behavior Model

In this section, we report our findings on employing our approach of creating user behavior models for each application.

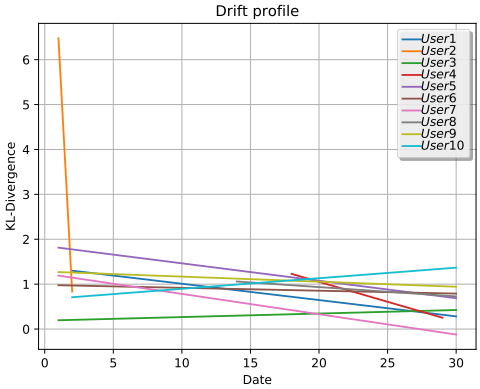
[1] https://phone-lab.org/experiment/data/
[2] https://github.com/UBOdin/jsqlparser
[3] https://github.com/UBOdin/PocketBench



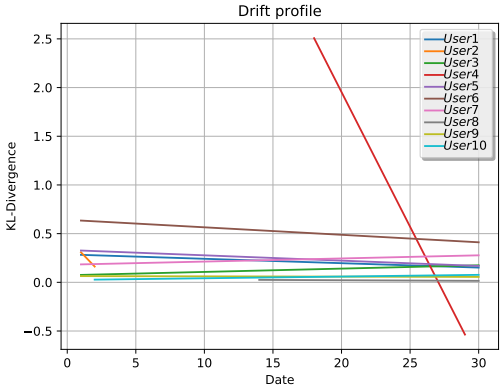
(a) Facebook



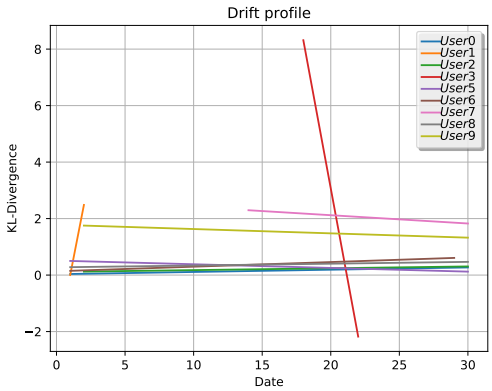
(b) Google+



(c) Hangouts



(d) Google Play Services



(e) Media Storage

Figure 6.2: Profile drift of each user for different Android applications

Figure 6.1 shows how user behavior changes over time. The x-axis represents the day of the month, the y-axis represents the drift score for the user for that specific day compared

to the aggregation of all activity in the previous days.

QMS allows its users to investigate reasons of the behavior changes by summarizing the features that caused the highest drift score change. This allows us to quickly inspect the information accessed when an alarm is raised.

Facebook application has 4 users, and except one, the users have stable profiles. One user, on the other hand, seems to have a distinguishable behavior change. However, when inspected, that specific user only uses the application more than 3 minutes twice, which explains the spikes seen in Figure 6.1a.

Similarly, except one user, all the users of Google+ application have steady behaviors. Most of the queries issued by Google+ application retrieve information on the user's account, which clearly shows how Android OS utilizes the Google+ application. Figure 6.2b, on the other hand, reveals that this application is mostly affected by the phone usage characteristics.

Hangouts, surprisingly, has the most adaptive user profiles. In Figure 6.1c, it can be seen that most user profiles have varying behaviors, which is understandable since all the other applications are used on a daily basis, but Hangouts application is a messaging platform which most people do not use everyday.

Google Play Services is an Android system-support application. The drift over time, as shown in Figure 6.1d, is low for most of the users. This application controls the install, update, and delete application operations on behalf of the operating system. The inspection we performed shows that the user who has a distinguishing behavior drift is used to install, and delete various applications.

Media Storage is another Android system-support application which mediates how the media files are stored on the phone. Again, except two users, the drift over time for most of the users is very small. However, there are two users who have very distinctive behaviors. When inspected, we saw that these two users use their phone as a music player.

Figure 6.2 shows the profile drift of each user for different Android applications. The x-axis represents the day of the month, the y-axis represents the expected drift score for the user for that specific day compared to the aggregation of all activity in the previous days. The trend line for each user represents the observed behavior drift, namely, how fast the behaviors of the user changes. Less area under the trend line means the user is less inclined to change their daily routines.

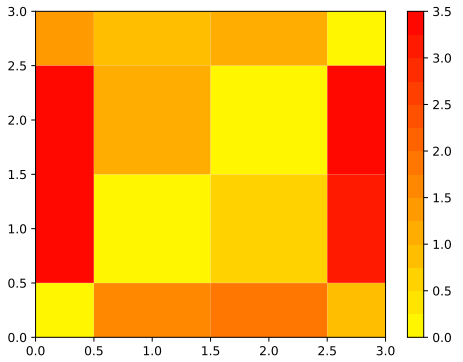
6.2.4 User Proximity

One misconception from Figures 6.1 and 6.2 can be that similar trends in these graphs do not mean these users have analogous behavioral characteristics. Similar trends in these graphs only mean that the expectation of behavior change pace is comparable for these users.

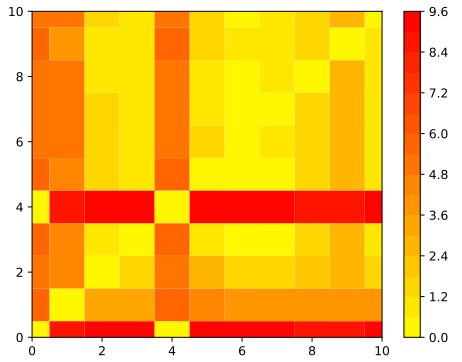
In this section, we investigate how similar the users profiles are to each other.

Figure 6.3 shows how comparable different users are with regards to their information access characteristics. In the graphs, darker colors represent that the corresponding user behavior is more distinct while lighter colors represents that the users have similar behaviors. Also note that the color scale is different for each application.

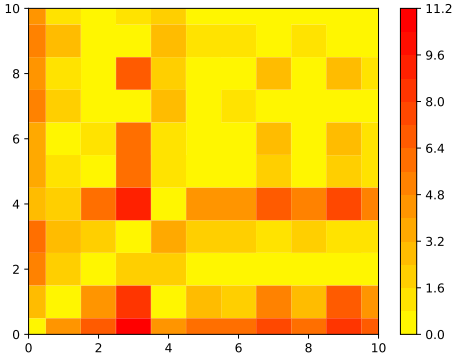
We observe that system services like usage characteristics of Google Play Services and Media Storage are analogous between users, whereas social media application usage characteristics of users are very distinguishable. In the following section, we will describe the red teaming approach we used to inject real workloads. Since these workloads were taken from the other users, the variety between users are directly correlated to the success of the experiments.



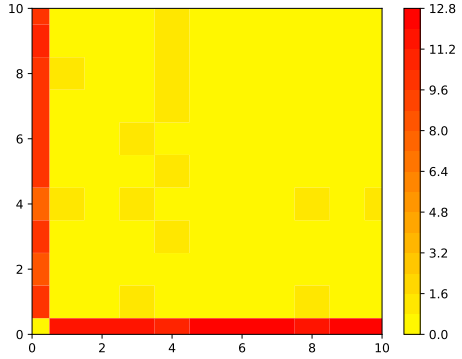
(a) Facebook



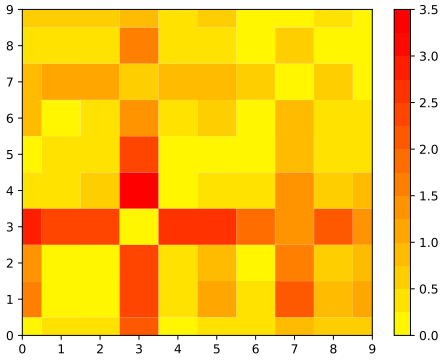
(b) Google+



(c) Hangouts



(d) Google Play Services



(e) Media Storage

Figure 6.3: Intra-Application user behavior difference for different Android applications

6.2.5 Red teaming approach

In this section, we describe the red teaming approach to validate our framework's effectiveness on identifying attacks. We consider two different attack frameworks: (1) Simulated query attack injection, in which we prepare specific attack scenarios for each application, and (2) Real workload injection, in which we input other users' real workloads into the user's own workload. We compare our system's true positive rate (TPR) against the false positive rate (FPR) to outlier detection model using various thresholds that do not consider temporal drift.

From now on, we will call the actual workload owner *the victim*, and the owner of the injected workload *the adversary*.

Simulated workload injection. We inject specifically designed workloads to perform a malicious activity into the user workload. We assume that all the actual query activity in the dataset is benign.

Note that we inject the simulated workloads into the log, not into the actual smartphone databases. Android OS does not let other applications to access another application's database, and it is possible that they have other defense mechanisms in place against attacks. The attack scenarios given here does not imply there is actually a vulnerability in the Android OS.

This approach addresses the privilege elevation and masquerade attack models described in Section 3.2:

1. In Android databases, the database tables are accessed by the application itself. When a new workload consisting of handwritten queries is injected and run, it would equate to a privilege elevation attack since normally the application wouldn't produce these queries therefore not letting the victim the privilege to access the information.

2. By injecting an adversarial workload, we simulate a masquerading attack. The adver-

sarial workload would use the victim's credentials. It can access and tamper sensitive data.

This approach also indirectly addresses the external attacks that use the victim's credentials to access sensitive data or elevate the victim's privileges to access more information. This would reflect characteristics of masquerading and privilege elevation attacks.

The simulated queries we injected in the victim's workload are prepared according to the scenarios given below:

Facebook We delete entries in the feed table and corresponding cache items. To replace them, we insert other feed items that we want the victim to believe.

Google+ We access the account information and the photos stored on the account. We alter the account information in order to redirect the password renewal emails to us.

Hangouts We access the locally stored messages and search for specific keywords to find sensitive information about the account owner. We insert the search results to the real time messages and send it to ourselves. We then remove the last message in order to cover our infiltration.

Google Play Services We modify the log records in order to confuse the operating system to skip updates for some applications. This would allow an adversary to take advantage of any patched vulnerabilities.

Media Storage We remove the metadata information of media files and replace them with other irrelevant information aiming to cause the operating system to crash.

The results for this approach are given in Table 6.3.

In our experiments, we created the normal user behavior model for each user. To compute detection accuracy rate, we partitioned the workloads of the same application created by all the other users day by day, and we injected simulated queries into the normal

Table 6.3: Detection rates for profile drift using simulated workload injection

		Threshold		Drift		LSTM	
	# of Attacks Performed	Detected	Success	Detected	Success	Detected	Success
Facebook	105	97	92.4%	98	93.3%	91	86.7%
Google+	225	202	89.8%	214	95.1%	203	90.2%
Hangouts	239	206	86.2%	206	86.2%	202	84.5%
Google Play	282	261	92.6%	267	94.7%	259	91.8%
Media Storage	282	251	89.0%	259	91.8%	249	88.3%

workload of users. As shown in Table 6.3, concept drift based methodology successfully determines these injections that do not belong to the user model at least 86% and at most 95% of the times depending on the application. We also set different thresholds for each application and reported the highest accuracy we gathered in the `Threshold` column of Table 6.3. Even when the ideal threshold set, drift based adaptive system performs better or equally in all cases. The LSTM model does not perform as well as the drift profile model in any of the cases, because the injected queries are not in the training set. Hence, the LSTM model labels these queries based on the closest user queries. Consequently, if the legitimate user queries are somehow more similar to the injected query than the queries issued by other users, the system fails to identify the injected query.

Real workload injection. We inject different workloads created by other users into the normal workload of the user. We assume that all the query activity in the dataset is benign. However, we simulate an attack by injecting one user's normal activity into the workload created by one of the other users. Hence, we only use queries that were created by the application itself.

This approach addresses the privilege abuse and masquerade attack models described in Section 3.2:

1. By injecting the adversary's workload, we simulate a masquerading attack. The adversary's workload would reflect its own characteristics while using the victim's cre-

Table 6.4: Detection rates for profile drift using real workload injection

		Threshold		Drift		LSTM	
	# of Attacks Performed	Detected	Success	Detected	Success	Detected	Success
Facebook	315	290	92.1%	283	89.8%	255	80.9%
Google+	2025	1817	89.7%	1818	89.7%	1892	93.4%
Hangouts	2201	1842	83.7%	1853	84.2%	1790	81.3%
Google Play	2583	2066	80.0%	2092	81.0%	2215	85.8%
Media Storage	2583	2099	81.3%	2105	81.5%	2301	89.1%

dentials.

2. We can also assume that the victim realizes that they have access to all of the database although they never needed to use it before, and start to access information that they don't need. This behavior also approximates to privilege abuse attack.

As we discussed in Section 3.2, this approach also addresses the external attacks that use the victim's credentials to access sensitive data, namely, the attack equates to masquerading attacks.

The results for this approach are given in Table 6.4.

In our experiments, we created the normal user behavior model for each user. To compute detection accuracy rate, we partitioned the workloads of the same application created by all the other users day by day, and we tested each of these partitions of data to see if our system raises an alarm on the specific normal user behavior model that is being tested. As mentioned in Section 6.2.5, although these injected workloads are actually naive workloads created by other users, in our concept, they belong to a different user; hence, they represent an extremely skillful attacker. As shown in Table 6.4, concept drift based methodology successfully determines these injections that do not belong to the user model at least 81% and at most 89.8% of the times depending on the application. We also set different thresholds for each application and reported the highest accuracy we gathered in the `Threshold` column of Table 6.4. Even when the ideal threshold set, drift based adaptive system performs

better except for Facebook app. Another interesting observation from the experiments is that the LSTM model performs better than the drift profile model in many cases, but it performs worse than the drift profile model for apps where the users tend to change their behavior patterns rapidly such as Hangouts app. This is due to the app logic creating similar query sequences, and the LSTM model not being able to determine where the injection starts and ends with real-world injections. In this case, many of the injected queries are still determined to belong to the legitimate user. Another observation is that the LSTM model performs better when there is more training data available, which is expected.

6.3 Summary

The focus of this chapter was to present a framework for creating user behavior profiles considering the temporal behavior drift to be used in data leakage detection from database systems. We first provided a user behavior model for data leakage detection. We argued that without considering the constant change in people's behaviors and habits, it is impossible for the defense systems to adapt to the new changes. This would result in the need for retraining of the user models for the system. In our experiments, we used real world query workloads and applied two different red teaming approaches: (1) simulated attack workloads, and (2) injection of benign real world workloads of other users. We evaluated our system with and without temporal behavior drift model in detecting these injections. We also investigated deep learning method LSTMs in terms of accuracy. These experiments yielded that the drift model performed consistently well in determining both handwritten attack queries and injected user workloads. The threshold based model performs comparably well when the ideal threshold is set, which is not guaranteed in real applications. The LSTM models performed consistently worse than both models in identifying handwritten attack queries, but performed better in most cases for injected user workloads.

In our experiments, we take the profile drift computation interval as one day, and we compare this distribution with the accumulated user profile over time. Although this approach achieves high detection rates in our setting, applying a sliding window on the streaming data to create the user profile can yield better results for different apps. The ideal length of the interval and sliding window depends on different settings.

Privacy concerns. Due to security and privacy concerns, it is almost impossible to collect real-world query logs from federal, state, or private organizations for research purposes. Even if the data shared is anonymized to a certain degree, to be able to keep the SQL queries intact for processing, the organizations still have to share sensitive information. The data shared can reveal the business logic of the information systems, database schema information, etc., which could leave them vulnerable to a data breach.

Chapter 7

Towards a Cyber Ontology for Insider Threats in the Financial Sector

Most security systems are built to protect sensitive information from external threats. Networked systems and information technology systems are changing with rapid innovations and they have been claiming more crucial roles in critical infrastructures. This rapid development has made the issue of information security more apparent due to the information contained in these systems.

Nowadays, insider threat is becoming an extremely serious security problem due to the threat it poses to the monetary assets and the sensitive customer data for financial institutions. The RAND report [5] addresses insider threat as “malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems." If there are not enough precautions taken, it can lead to insider attacks.

The banking domain differs from the others in the sense of information security. Most of the employees like tellers and customer representatives in a banking institution have the ability to access very sensitive information. This makes performing an insider attack

in banking organizations easier. An attack that is coming from an employee within an organization can go unnoticed for a very long time [1] and the implications can be very costly.

Examining the chronology of events is not enough to understand if the intent is malicious. Insiders usually have authorized access to the information they target, and their behaviors are very difficult to distinguish from normal activities, which leads to false negatives. On the other hand, a benign intent can sometimes be observed as malicious behavior. Acting upon a false positive without evaluating the sequence of events and their details carefully can lead to a message of distrust between the employees and the administration in an organization.

Even though there are other data preservation techniques and systems, relational databases are heavily used in back-end servers to store financial data, which consists of a lot of sensitive information. This makes relational databases a primary target for cyber crime and identity theft. The aim of this effort is to develop an ontology of this area, which is expressed in the Web Ontology Language (OWL), so that it ensures integration with other knowledge domains and enables data integration across different data sources. Semantic web applications are becoming more popular by the day and ontology is the most important enabling technology of these applications. Basically, it describes terms and different relationship types between terms. In this chapter, we create a taxonomy of insider threats and identify the relationships between the entities we define in the taxonomy. These entities and relationships are used to create an insider threat ontology which is then mapped onto upper ontologies and domain ontologies that are commonly used in financial systems. We use this technology to make background knowledge of the domain explicit for computers, so that for context we feed, the system can “understand" and “reason with" the information as humans do.

The ontology uses first order logic to determine if an event leads up to a malicious

activity by using the relationships between objects. The events that are determined to be malicious are reported to the security experts, who evaluate the report to make a final decision. However, there are many other relationships and rules that can be discovered and learned. We feed the labels provided by the security experts back to the system to create an active learning mechanism. The negative feedback is then used to construct new rules to prevent similar events from raising false alarms.

The contributions of this chapter are providing support to the security experts in the investigation process; creating a framework of a cyber ontology for insider threats in the financial sector focusing on relational database management systems; integrating this ontology with commonly used ontologies the Suggested Upper Merged Ontology (SUMO) [74, 77], Friend of a Friend (FOAF) [30, 40] and Finance [100] to make it applicable and integrable to the systems that use these ontologies; and finally, exploiting the capabilities of ontologies, providing a methodology to distinguish between benign and malicious event when a suspicion arises.

There are several challenges when developing such a framework. Mainly due to regulatory reasons, financial institutions cannot share sensitive information which includes their internal structure. This results in uncertainty on analyzing what data is available to us and shaping how to validate our initial proof of concept. The other major challenge is correctly building the ontology. The representation of the world should be accurate and precise. Inaccurate and insufficient analysis of the domain creates risk of misrepresenting the domain.

Section 7.2 explains the required information to create the knowledge base and for the evaluation of the ontology. Section 7.3 performs the validation of the ontology based on insider attack scenarios given. Section 7.4 discusses the advantages and contributions of our research.

7.1 Methodology

Taxonomy and ontology are two common terminologies that are being used in information management and there are cases where people treat them as synonyms.

The term “taxonomy" could refer to a hierarchical classification or categorization system, or to an organization of concepts of knowledge, as well as a knowledge organization system designated to include term lists and classifications [43]. Except for some rare cases, defining the relationships between entities is not a concern when defining taxonomies, other than a hierarchical relationship between entities.

The term “ontology" but for its philosophical meaning, is a formal framework to represent knowledge in computer and information science. Ontologies define classes, properties of these classes and relationships between these classes within their domain. Using the relationships, we can extract other information from these information entities and use them to identify other previously unidentified relationships between them. The authors of [33] classify taxonomies as linguistic/terminological ontologies. However, taxonomies can also be used to define ontologies when the relationships between the classes are defined and a formal structure of an ontology can be constructed with them. How to develop an ontology is summarized in [75] as

- Defining classes in the ontology
- Arranging the classes in a taxonomic hierarchy
- Defining slots and describing allowed values for these slots, namely, creating properties of the classes
- Filling in the values for slots for instances.

We model the normal world, not the insider threat, while creating the ontology framework for this study, and use normality to form the story of the attack. We capture behavior

anomalies, using the SQL queries that are issued on the databases of the organization, which is not in the scope of this work. By profiling user behavior in an RDBMS to detect anomalies, we can mark some queries as suspicious [65] and trigger other mechanisms to investigate what the real intent is. As shown in Figure 7.1. the scope of this work is within the circle. When we capture a suspicious intent, we fill the ontology slots for instances with the event logs that are also stored by the organization. These logs include most of the behaviors that can be classified as normal behavior, but exploiting the capabilities of ontologies, we can query the ontology and capture the anomalies, so that we can classify the suspicious intent. The suspicious intent can be classified as benign, or it can be classified as one of the insider attack types. The duty of the management of the organization or the security expert is inspecting the instances and connections in the ontology when it decides that it is an insider attack, and then building the story or with reverse engineering techniques.

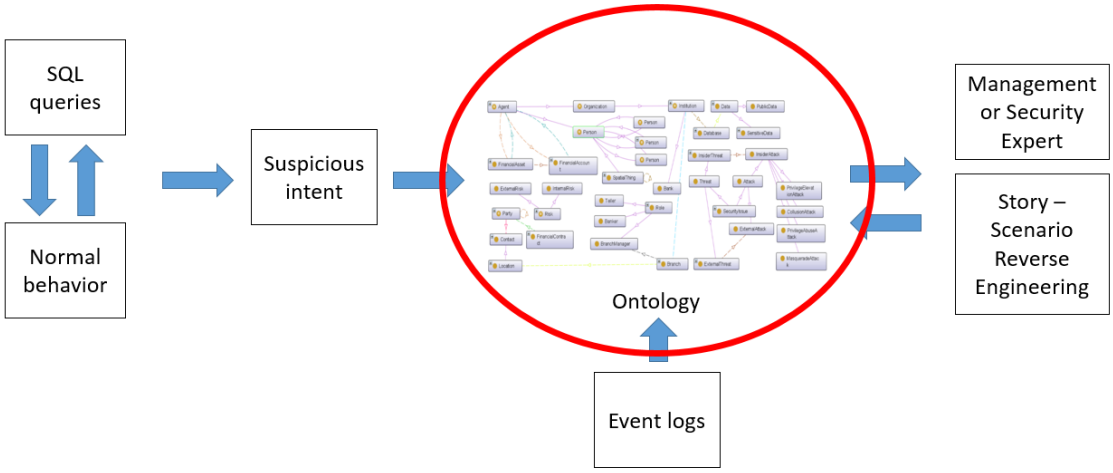


Figure 7.1: Insider threat identification scheme

This section identifies the methodology employed in the taxonomy and ontology development process and explains the details of the construction of ontology classes.

7.1.1 Ontology development

The ontology development process we employed in this chapter is a top-down analysis which requires understanding the semantics of the end-users who will actually use the resulting ontology. It starts with creating a list of terms which will be used to construct the taxonomy of the structure. The taxonomy needs to include the terms that define the classes in the domain. The taxonomy needs to be limited with what the resulting ontology will cover, what will it be used for, and what types of questions the ontology will answer to. Following the creation of the taxonomy, and the hierarchy within the taxonomy, the properties of the classes should be defined along with the relationships between classes.

Feedback. The development of the ontology requires input from many actors and experts. In a banking environment, there are many rules, processes, and variables that should be considered. Hence, the ontology development process is challenging, and it can become a tedious task. Instead, it is possible to create a base ontology, and then extend it with the input provided by the anomaly detector using lexical learning [60].

The ontology-based system takes its input from the anomaly detector, and it provides three possible outputs: (1) malicious activity, (2) benign activity, and (3) undetermined. The outputs are then presented to the security experts, who evaluate the output and give feedback to the ontology-based system if there is a mistake as seen in Figure 7.2. The ontology-based system uses this information to extend the ontology rules using lexical learning. As a result, when the system encounters a similar anomaly, it does not raise a false alarm.

Validation. The validation of the ontology structure is performed through competency questions. These questions assure that the targeted value of the structure is achieved. They serve as procedures that indicate when the ontology development is sufficiently complete. The competency questions aim to ensure that the results are accurate, sufficient, and has the

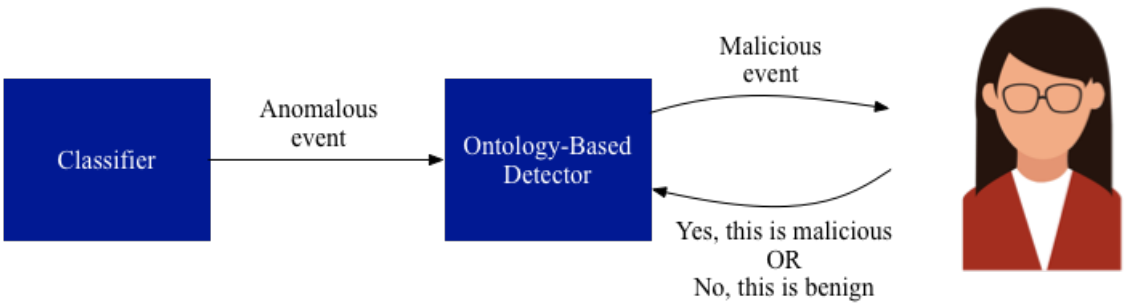


Figure 7.2: Feedback mechanism

right level of granularity which is identified by the subject matter expert. They also ensure that the scope of the ontology is still within the limits.

Integration. It is essential to integrate the ontology created with other ontologies, as it will integrate the domain with the rest of the world. Considering that ontologies are a web of knowledge, integrating the ontology with other ontologies will create a bigger knowledge base and extend the opportunities of integrating this ontology with the existing systems. However, to increase data and information quality within a domain, we need to create an ontology that can represent that domain successfully, and creating an ontology requires expert knowledge within that specific domain as well as the skills required to create it. To create an ontology, ontology developers and domain experts need to work together. Ontologies that are created by people who lack either expert knowledge or ontology development skills may result in serious problems and wrong results. However, not all research projects have enough resources to hire people who have these skills. Also, even if the resources are sufficient, project teams may not think it is necessary.

7.1.2 Taxonomy

The efforts we have put into creating a taxonomy on finance domain has resulted with the taxonomy shown in Figure 7.3. As a result of the top-down analysis we performed with the domain experts of our collaborator banking institution, we have identified the taxonomy

classes based on basic scenarios given in Section 7.3. The validation of these classes is made through mapping between classes and instances gathered from the mentioned scenarios.

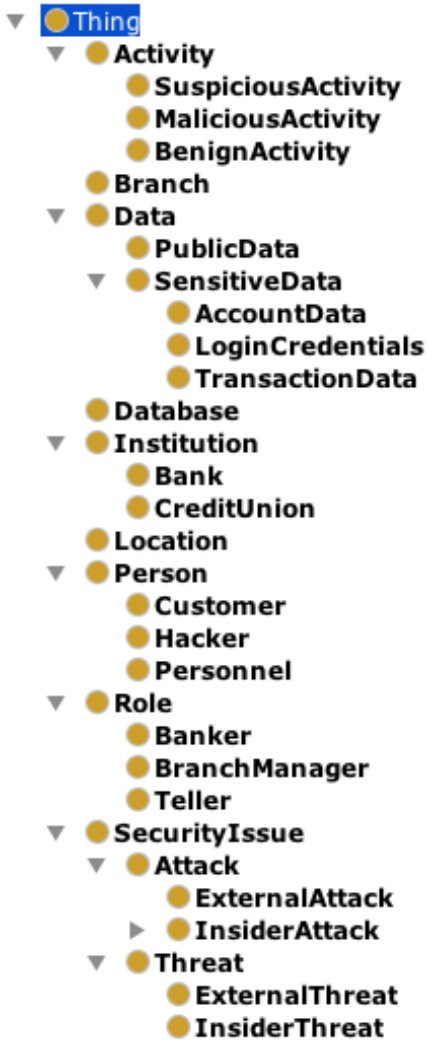


Figure 7.3: Ontology classes created from initial terminology

7.1.3 Ontology

There are several types of ontologies that we can base the rules of our ontology framework.

Upper level ontology. The ontologies that belong to this level describe concepts that are the same across all knowledge domains which provides a high level of semantic interoper-

ability.

Domain ontology. The ontologies that belong to this level describe concepts in a specific field or in a part of the world. This specific field or part of the world represents the domain that the ontology describes. Since the concepts belong to the domain, they may or may not be compatible with a concept that has the same name in a different domain ontology.

Hybrid ontology. The ontologies that belong to this level describe concepts that can be both mentioned in domain and upper level ontologies. Especially by working on integration of different systems together, the hybrid approach makes it easier to work with multiple ontologies. Some concepts can be defined universally but some concepts are described according to the domain related limitations.

Our goal is to provide a web of knowledge by integrating commonly used upper ontologies into our ontology. To achieve this task, we create a domain ontology on insider attacks focusing on financial sector, and then we identify some ontologies that are commonly used by academia and industry that may possibly have similar classes that we identified in our ontology.

Friend of a Friend (FOAF) ontology [40] [30] describes people, their activities and relationships between each other and other objects. It allows groups of people to create social networks, which we are using to describe the relationships between customers, bank personnel and roles and hierarchy within the organizations. The common terms that we import from this ontology are “Organization" and “Person" classes as can be seen in Figure 7.4. After importing these classes, we expand this terms with the domain specific subclasses, to define the banking environment.

The Suggested Upper Merged Ontology (SUMO) [77], has a broad range of domain areas included in it. However, it only provides a structure and a set of general concepts upon which domain ontologies could be constructed. Financial concepts are among these concepts, too. The common terms that we import from this ontology are “FinancialAc-

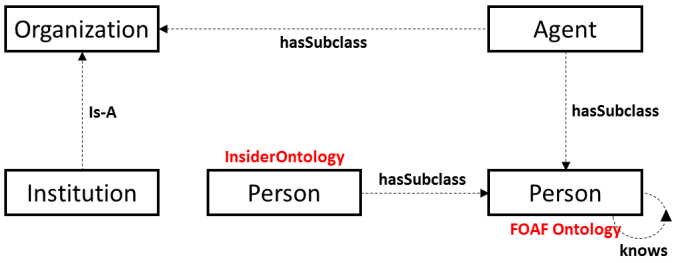


Figure 7.4: Integration of FOAF ontology classes

count", “FinancialContract", “financial asset" and all of their subclasses. The relationships that these terms have with the other classes in our ontology can be seen in Figure 7.5.

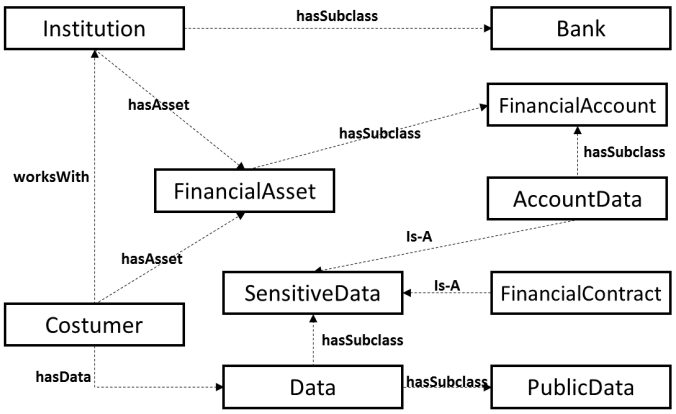


Figure 7.5: Integration of SUMO ontology classes

Finance ontology [100] is an ontology on financial instruments, involved parties, processes and procedures in securities handling. We are using this ontology to define the financial instruments and involved parties within organizations, so that we do not recreate an already modeled structure and the main concern of our ontology stays as usual banking processes and insider attacks. The common terms that we import from this ontology are “Address", “Party" and all of the subclasses of “Party" as can be seen in Figure 7.6.

Therefore, we integrate our ontology with FOAF to base our Person and Organization structure on universally defined terms and we expand these terms. On the other hand, we import classes from SUMO and Finance ontology to use the classes that are already defined in financial domain, so that we do not need to define new classes in the finance domain. The graph of the main components of resulting ontology is shown in Figure 7.7.

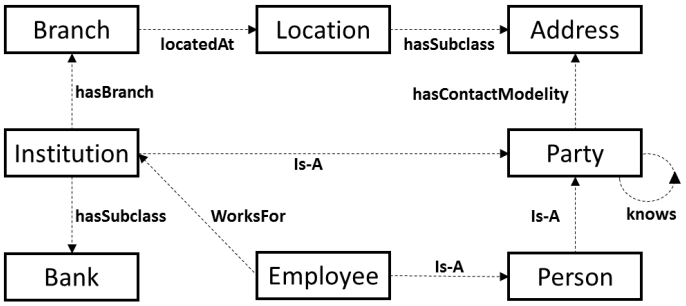


Figure 7.6: Integration of Finance ontology classes

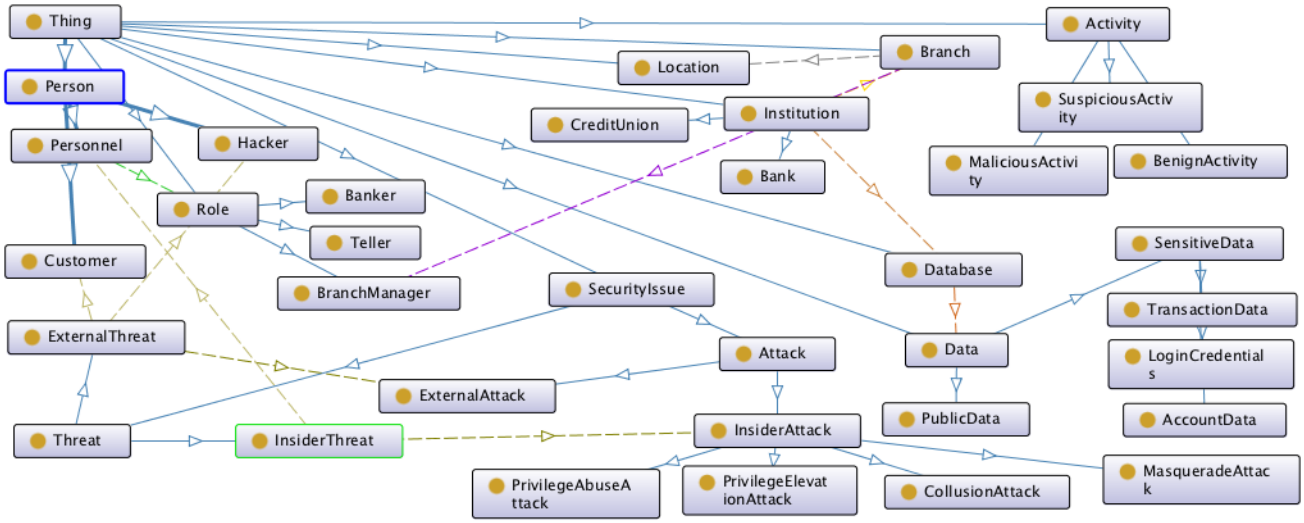


Figure 7.7: Main components of the ontology

7.2 Data Sources

Most institutions cannot provide or reveal all the details of insider attacks due to regulatory reasons. Therefore, we validate our initial proof of concept based on the limited data that we are provided with by the collaborating financial institution.

7.2.1 Databases and log files

We need to understand access patterns for a database in a financial institution database system. This includes a snapshot of the data in the database, as well as query logs including:

- The look up or update query being issued

- (Anonymized) identification of the user or role that accessed the data
- (Anonymized) physical or IP address of the machine issuing the query

What we try to achieve here is to see a view of the daily life of that database. The size and traffic information of the database can be gathered from the log entries. However, we don't have access to the database itself, or its snapshot.

7.2.2 Organization and permission structure

To have a better understanding of the internal structure of the environment, we create a taxonomy to identify roles and differentiate between different attack types. This taxonomy for the user roles includes the following information:

- User roles
- Access Permissions
- Manages
- Managed by

For instance, what does a bank teller do? What privileges does a teller have on the database? Who manages the teller and who has more privileged access to the system in that branch? Who can change the privileges given to a user? We need to ask these questions for selected types of users. Some of the relationships that are represented in the ontology does not have to present in order to trust the resulting structure. This missing values can be created in time, or left blank.

7.2.3 Normal behavior

Experimenting on existing logs of a database reveals a lot of information to profile the normal behavior of users. However, taking this data as ground truth means that we accept insider attacks that have not been identified as normal behavior. Hence when we encounter such characteristics, we may classify the intents of those behaviors as benign. As stated before, the ontology structure aims to model the normal world, not the attacks. We use the logs of databases to create normal behavior patterns which then should be reviewed by banking experts.

7.2.4 Previously identified attacks

We will use attacks identified in the past to create new insider attack scenarios and simulate them on the example databases. Real examples provided by financial institutions will be a guideline preparing these scenarios. The data, log files, and scenario details about some insider attacks that are detected before should be very useful in this phase.

These scenarios can either be used to *create attack models*, or *simulate an attack* to see if the system accurately identifies an insider attack. Although this information would be very beneficial if it existed in order to see how a real attack is represented in the ontology, the experts of the institution we are collaborating expressed that this information is one of the most classified data types.

7.3 Validation

The insider threat ontology includes the following insider attack types: masquerade attacks, privilege elevation attacks, privilege abuse attacks and collusion attacks. Some attacks may appear in various cases and they may seem very different than each other even if they

belong to the same classification, some can even happen unintentionally. According to our study, the classification given above covers all types of insider attacks reported in the literature. If novel attacks are later discovered, additional types can be easily integrated into the ontology framework.

The validation we perform in this chapter is made through insider attack scenarios and it aims to test the competency of the ontology in representing any attack that an organization can experience. These scenarios are based on general banking processes and they are not based on real attacks, since we are not provided with that information. To perform this validation, we use description logic (DL) to show how the ontology deals with the information that is provided by the information systems. We choose SROIQ [45] which is considered one of the most expressive DLs and can be considered as a syntactic variant of OWL DL [53]. This DL basically has two kinds of clauses:

- T-Box: it describes the rules of the world as they are described in the knowledge base, and only includes the relationships between classes. For example:

$$employee \sqsubseteq person \quad \text{“employee is a person"}$$

$$customer \sqsubseteq person \quad \text{“customer is a person"}$$

- A-Box: it describes the properties of individuals and their relationships. For example:

$$person(mary) \quad \text{“mary is a person"}$$

$$person(matthew) \quad \text{“matthew is a person"}$$

$$loves(mary, matthew) \quad \text{“mary loves matthew"}$$

We provide a step by step process for each type of attack. For each step, we show

- Plain text: what relevant information we can have from the information systems represented in plain text form.
- Description logic: formal representation of the information we have in the corresponding step.
- Data source: what sources we use while compiling this information.
- Comment: additional explanations if need.

The following part explains the above four attack types and provides an example each.

7.3.1 Masquerade attacks

In a masquerade attack, the attacker illegitimately assumes the identity of a legitimate user [84]. Before launching an attack like this, the attacker must gather the credentials to access the system. However, the gathering phase of the attack is outside of the scope of this chapter, and so we will make the assumption that the attacker has already gained the credentials necessary to access the system. Here, it is clear that the attackers try to hide their identity and make the victim responsible for any action they take. An example scenario is given below and the analysis of the scenario is given in Table 7.1.

Scenario: Amelia and Ben are tellers and work at the Amherst branch of a bank. Cecile comes to the branch and asks Amelia for her account balance and withdraws some money from her account. Amelia steps out to a meeting with the branch manager and leaves her computer (PC1) logged in to the system thinking that she will be gone for a very short time. Ben takes advantage of her absence, and uses her computer to look up a Dan and Emily's social security numbers and transactions to use them later. He returns back to his seat before Amelia comes back. A few days later, Dan realizes that there is a credit inquiry in his credit report and contacts the bank about the source of the inquiry.

T-Box rules are:

$$Customer \sqsubseteq \exists hasData.CustomerData$$

$$\exists checks.CostumerData \sqsubseteq Employee$$

$$checks \sqsubseteq knows \circ hasData$$

which mean “Every Customer has CustomerData", “Only Employees check customers’ data", and “If you check someone’s data, you must know that person," respectively.

The other T-Box rules are not needed in the given scenario, hence they are not given.

Table 7.1: Masquerade attack

#	Plain text	Description logic	Data source	Comment
1	Amelia is a person and a teller	person(Amelia) employee(Amelia) teller(Amelia)	Organizational Structure	
2	Ben is a person and a teller	person(Ben) employee(Ben) teller(Ben)	Organizational Structure	
3	Amelia works at Amherst Branch	branch(Amherst) worksAt(Amelia, Amherst)	Organizational Structure	
4	Ben works at Amherst Branch	worksAt(Ben, Amherst)	Organizational Structure	
5	Amelia works with Ben	worksWith(Amelia, Ben)	Organizational Structure	

6	Ben works with Amelia	worksWith(Ben, Amelia)	Organizational Structure	
7	Amelia uses PC1	equipment(PC1) uses(Amelia, PC1)	Organizational Structure	
8	Ben uses PC2	equipment(PC2) uses(Ben, PC2	Organizational Structure	
9	Cecile checks into Amherst Branch	visits(Cecile, Amherst)	Query Logs	Cecile swipes her card to confirm her identity
10	Cecile is a customer of Amherst Branch	person(Cecile) customer(Cecile) customerOf(Cecile, Amherst)	Database	Cecile's information is retrieved from database
11	Amelia confirms Cecile's identity	knows(Amelia, Cecile)	Query Logs	Interaction creates a relationship between Amelia and Cecile

12	Amelia looks up Cecile's data	account(Cecile's data) hasAccount(Cecile, Cecile's data) checks(Amelia, Cecile's data)	Query Logs	Important point here is that these queries follow the last 2 steps
13	Amelia checks balance for Cecile	checks(Amelia, Cecile's data)	Query Logs	
14	Amelia withdraws money for Cecile	changes(Amelia, Cecile's data)	Query Logs	At the end of this operation, Amelia leaves and Ben uses the computer but our systems don't know it

15	Amelia looks up Dan's data	customer(Dan) account(Dan's data) hasAccount(Dan, Dan's data) checks(Amelia, Dan's data)	Query Logs	Suspicious query should be caught by the system and trigger the ontology, because this is not normal behavior of Amelia.

16	Amelia looks up Emily's data	customer(Emily) account(Emily's data) hasAccount(Emily, Emily's data) checks(Amelia, Emily's data)	Query Logs	Suspicious query should be caught by the system and trigger the ontology, because this is not normal behavior of Amelia. At this point, the events are under investigation.
17	Dan makes a credit inquiry		Query Logs	
18	Dan appeals to the bank		Query Logs	The required actions should take place here according to the result of the investigation

With the T-Box rules given above, the DL requires knows(Amelia, Dan) and knows(Amelia, Emily) which don't exist. The activities in Step 15 and Step 16 violate the rules, hence

make the T-Box rules unsatisfiable.

7.3.2 Privilege elevation attacks

In a privilege elevation (also known as privilege escalation) attack, a user with insufficient permissions accesses the information that only a more privileged user can see. The attackers usually exploit a vulnerability of the system to escalate granted permissions [29], so that they can use these new permissions to access information. An example scenario is given below and the analysis of the scenario is given in Table 7.2.

Scenario: Ben is a teller at the Amherst branch of a bank. He finds out that after a software update the system allows him to see all of the sensitive information of the bank's customers and thinking that he is allowed to see them, he doesn't notify his superiors. Ben uses these privileges to look up the other teller Dan's sensitive information out of curiosity. However, he does not take things further.

T-Box rules are:

$$Customer \sqsubseteq \exists hasData.CustomerData$$

$$Teller \sqsubseteq \forall checks.CustomerData$$

which mean "Every Customer has CustomerData", and "Every Teller can only check CustomerData," respectively.

The other T-Box rules are not needed in the given scenario, hence they are not given.

Table 7.2: Privilege elevation attack

#	Plain text	Description logic	Data source	Comment

1	Ben is a person and a teller	person(Ben) employee(Ben) teller(Ben)	Organizational Structure	
2	Ben works at Amherst Branch	branch(Amherst) worksAt(Ben, Amherst)	Organizational Structure	
3	Ben works with Dan	person(Dan) employee(Dan) worksWith(Ben, Dan)	Organizational Structure	
4	Ben uses PC1	equipment(PC1) uses(Ben, PC1)	Organizational Structure	

5	Ben looks up Dan's data	account(Dan's data) hasAccount(Dan, Dan's data) checks(Ben, Dan's data)	Query Logs	Suspicious query should be caught by the system and trigger the ontology, because this is not normal behavior of Ben. Ben looks up an employee's information for the first time. The event will violate the given T-Box rule that specifies a teller can only check customer data.

With the T-Box rules given above, the DL requires Dan to be a customer, not an employee. The activity in Step 5 violates the rules, hence makes the T-Box rules unsatisfiable.

7.3.3 Privilege abuse attacks

In a privilege abuse attack, the user uses his/her permissions to retrieve information that he/she has no need to know. An example scenario is given below and the analysis of the scenario is given in Table 7.3.

Scenario: Ben is a branch manager of the Amherst branch at a bank in Buffalo, NY. He also looks up sensitive information of some people from New York, NY. As not to look suspicious, he chooses the customers of a specific branch, keeping in mind that people from the same household tend to travel together and have bank accounts from the same branch.

T-Box rules are:

$$Customer \sqsubseteq \exists hasData.CustomerData$$

$$Customer \sqsubseteq \exists visits.Branch$$

$$Employee \sqsubseteq \exists worksAt.Branch$$

$$hasData \circ checks^{-1} \sqsubseteq visits \circ worksAt^{-1}$$

which mean “Every Customer has CustomerData", “Every Customer visits a Branch", “Every Employee works at a Branch", and “You can only check someone’s data, if the owner of the data visits the Branch you work at," respectively.

The other T-Box rules are not needed in the given scenario, hence they are not given.

Table 7.3: Privilege abuse attack

#	Plain text	Description logic	Data source	Comment
1	Ben is a person and a branch manager	person(Ben) employee(Ben) branchManager(Ben)	Organizational Structure	

2	Ben works at Amherst Branch	branch(Amherst) worksAt(Ben, Amherst)	Organizational Structure	
3	Ben looks up for Dan, a customer of a branch in New York, NY	person(Dan) customer(Dan) account(Dan's data) hasAccount(Dan, Dan's data) checks(Ben, Dan's data)	Query Logs	Suspicious query should be caught by the system and trigger the ontology based system, because this is not normal behavior of Ben. The event will violate the given T-Box rule that specifies the requirement of the customer should visit the branch that the employee works at.

With the T-Box rules given above, the DL requires Dan to visit the branch where Ben works at. The activity in Step 3 violates the rules, hence makes the T-Box rules unsatisfiable.

7.3.4 Collusion attacks

In a typical collusion attack, there are usually more than one people with different privileges collaborating to access and harvest information [90]. Since this data is usually supposed to include more relations and be more extensive, the impact of these attacks is usually higher. An example scenario is given below and the analysis of the scenario is given in Table 7.4.

Scenario: Carl is a branch manager and Karen is a secretary at the same branch of a bank. Carl leaks information from the database systems and from internal documents of the bank to a rival company. However, after gathering them, Carl hides the information along with a lot of other information. He orders Karen to collect some specific information and send it to a specific addresses after adding it to the files that he gave her. Karen doesn't know that she is collaborating but she doesn't check out what she is sending.

T-Box rules are:

$$SuspiciousActivity \doteq BenignActivity \sqcup MaliciousActivity$$

$$BenignActivity \sqcup MaliciousActivity \sqsubseteq \bot$$

$$SuspiciousActivity \subseteq \exists mayLeadTo.InsiderThreat$$

$$MaliciousActivity \subseteq \exists leadsTo.InsiderAttack$$

which mean "SuspiciousActivity is either a BenignActivity or MaliciousActivity", "BenignActivity and MaliciousActivity are two disjoint types", "Every SuspiciousActivity may lead to an InsiderThreat", and "Every MaliciousActivity leads to InsiderAttack," respectively.

The other T-Box rules are not needed in the given scenario, hence they are not given. The important point in this scenario to realize is that this time there is no inconsistency

in the knowledge base. The system just marks an activity as SuspiciousActivity and it is reported to the security unit.

Table 7.4: Collusion attack

#	Plain text	Description logic	Data source	Comment
1	Carl is a person and a branch manager	person(Ben) employee(Ben) branchManager(Ben)	Organizational Structure	
2	Karen is a person and a secretary	person(Karen) employee(Karen) secretary(Karen)	Organizational Structure	
3	Carl works at Amherst Branch	branch(Amherst) worksAt(Carl, Amherst)	Organizational Structure	
4	Karen works at Amherst Branch	worksAt(Karen, Amherst)	Organizational Structure	
5	Carl looks up for daily transactions	sensitiveData(dailyTransactions) checks(Carl, dailyTransactions)	Query Logs	This won't be a suspicious query because as a branch manager, Carl is expected to monitor daily transactions

5	Karen looks up for daily branch statistics	sensitiveData(branchStatistics) checks(Karen, branchStatistics)	Query Logs	This won't be a suspicious query because as a secretary, Karen is expected to report on this data. However, we expect our anomaly detection system to identify the harvesting. In that case, the ontology based system is triggered. The system creates a SuspiciousActivity individual in the ontology and reports consecutive access to sensitive data which indicates collusion.

It is important to notice that all attack types has a characteristic violation. Depending on the violations, the ontology can identify the attack type with the rules using the same methodology.

7.4 Discussion

We have presented an ontology framework focusing on insider attacks in banking domain targeting database systems [55]. We extended our work providing example scenarios on different attack types and justifying how such a system can be useful against insider threats. As indicated before, the prior efforts in insider threats branches to two different directions: psychological aspects and physical aspects of insider threats. Our chapter extends the work on building a cyber ontology for insider threats in the financial sector, as it is critical to having capable systems that can identify such threats and developing countermeasures against insider attacks in this domain. The specific contributions of our chapter are,

- making use of ontologies as a supporting mechanism to validate the outcome of other anomaly detection systems
- creating a cyber ontology framework for insider threats in the financial sector focusing on relational database management systems
- providing support to the security experts in the investigation process
- ensuring the integration with other knowledge domains to enable data integration.

The literature survey we have conducted shows us that this ontology fills the gap in ontological structuring of insider threat research in the financial sector. The ontologies

developed on insider threat research generally focus on defining what an insider threat is and how it takes place [26, 70]. The work in [26] leads us to experiment on specific domains and use the domain specific knowledge to create a semantic structure while the ontology proposed in [70] shows the relationships between insider threat concepts and insider threat activity. This structure defines the insider threat in financial sector more conclusively. Even if we have collaborated with financial sector experts, we know that there is still a lot to do to expand the capability of our ontology, since we are restricted to gather real data from banking databases. One more way of validating our ontology and consistency of our system is to create a synthetic database. However, to create a synthetic database, we still need to have statistical information of the database tables and columns which is not readily available for us.

The cyber ontology we created has classes from FOAF and SUMO ontologies, which are universally defined, and the terms in them mean the same across all knowledge domains. In this sense, our ontology provides a high level of semantic interoperability. When fully developed, we believe that this integration with other domains and semantic structures approach can prove effective to addressing more factors about insider threats as it could be used by researchers to test and evaluate their detection and mitigation schemes, as well as identifying similar attacks by using previously identified attacks. The human-in-the-loop feedback mechanism extends the ontology developed, and adapts it to the processes in the organization. Consequently, it reduces false positives in time.

The validation scheme we performed is using the same ontology relations and classes. We represent ontology instances in description logic and show that how the ontology captures the differences between different attacks. However, the performance of this system completely depends on another system that profiles user behavior on the database system and catches anomalies in the user behavior depending on the SQL queries that the user sends to the system.

Chapter 8

Conclusion

Current insider threat detection mechanisms usually depend on detecting anomalies in a user's behavior, focusing on a specific type of resource, or a combination of resources [83]. These resources can be shell commands [73], file accesses [68], and SQL queries [48, 64].

In this dissertation, we focused on insider threats against relational databases. The literature clearly lacked accurate modeling of insider threats in this domain, an analysis of computational complexity for user intent modeling, and a generalized strategy for dealing with such attacks. We bridge this gap by constructing the notion of user intent model, and creating a generalized threat model for database systems.

We then constructed a framework for modeling normal behavior of users on database systems for detecting anomalous behavior while considering temporal behavior drift. This model enables us to address insider attacks such as masquerading, privilege escalation, and privilege abuse, as well as some cases of external attacks that compromise legitimate user accounts. In this model, first, a SQL query feature extraction scheme is constructed. Then, an adaptive user behavior modeling scheme that is usable in any database management system is developed. Finally, we showed how anomalous behaviors can be detected while considering the temporal drift in user behavior. We modeled the change in user behavior

by comparing probability distributions over time, and determined the expected behavior change through linear regression for each user profile. We also developed a deep learning model to compare the performance of these models. We validated our model using a real world database query workload of various smartphone applications.

We also presented an insider threat ontology, and a methodology that it can be used in order to distinguish benign outliers and malicious activities. To create the ontology, we used a top-down analysis approach to define a taxonomy and identify the relationships between the taxonomy classes. The ontology is mapped onto the Suggested Upper Merged Ontology (SUMO), Friend of a Friend (FOAF) and Finance ontologies to make it integrable to the systems that use these ontologies and to create a broad knowledge base. It captures masquerade, privilege elevation, privilege abuse and collusion attacks and can be extended to any other novel attack type that may emerge. It classifies an attack using the knowledge base provided and the missing relationships between classes. The ontology can be extended using human-in-the-loop feedback mechanisms by using lexical ontology learning to adapt to the needs of the organization, which eliminates the need to create a complete ontology model to start using this system.

8.1 Future Work

We will consider techniques on extending anomaly-based insider detection techniques to rely on temporal behavior drift as well as identifying the areas the user might be interested in by analyzing the user's activity history. This will help us detect attacks by determining the anomaly severity not only with a score, but also with the context of the anomaly. The current anomaly-based detection mechanism uses linear regression model. We will test other regression models with the temporal behavior drift to test the performance of the system.

Our work can also be seen as a guideline to improve blockchain technology to create a verifiable database system that allows the data owners to store their databases without worrying that an attack to the storage server or communication channel can manipulate the data on the server. The current state of the art technology, which addresses data manipulation attacks to the storage server, relies on authenticated data structures and does not support some SQL operations. This approach may have the potential to eliminate database system dependency and the need to modify the underlying database system completely.

Bibliography

[1] The insider threat: An introduction to detecting and deterring an insider spy. http://www.fbi.gov/about-us/investigate/counterintelligence/the-insider-threat. Accessed: 2015-06-15.

[2] R. Agrawal, P. Bird, T. Grandison, J. Kiernan, S. Logan, and W. Rjaibi. Extending relational database systems to automatically enforce privacy policies. In *ICDE*, 2005.

[3] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *ACM SIGMOD*, 2006.

[4] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measures for OLAP sessions. *Knowledge and information systems*, 2014.

[5] R. H. Anderson and R. Brackney. Understanding the insider threat. Santa Monica, CA, USA, March 2005. RAND Corporation. Also available in print form.

[6] K. Aouiche, P.-E. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In *ADBIS*, 2006.

[7] M. A. Babin and S. O. Kuznetsov. Recognizing pseudo-intents is conp-complete. In *CLA*, 2010.

[8] S. Bandhakavi, P. Bisht, P. Madhusudan, and V. Venkatakrishnan. Candid: preventing sql injection attacks using dynamic candidate evaluations. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 12–24. ACM, 2007.

[9] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.

[10] M. Bishop, S. Engle, D. A. Frincke, C. Gates, F. L. Greitzer, S. Peisert, and S. Whalen. A risk management approach to the "insider threat". In *Insider threats in cyber security*, pages 115–137. Springer, 2010.

[11] M. Bishop and C. Gates. Defining the insider threat. In *CSIIRW*, 2008.

[12] S. W. Boyd and A. D. Keromytis. Sqlrand: Preventing sql injection attacks. In *International Conference on Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[13] D. F. Brewer and M. J. Nash. The chinese wall security policy. In *Proc. of the 1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE, 1989.

[14] C. Brodsky. Database activity monitoring(dam): Understanding and configuring basic network monitoring using imperva's securesphere. *Sans Institute InfoSec Reading Room*, 2015.

[15] N. Bruno and S. Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In *ACM SIGMOD*, 2005.

[16] CERT Insider Threat Center. 2014 U.S. State of Cybercrime Survey. July 2014.

[17] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[18] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.

[19] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[20] B. Chandra, B. Chawda, B. Kar, K. V. Reddy, S. Shah, and S. Sudarshan. Data generation for testing and grading SQL queries. *VLDBj*, 2015.

[21] B. Chandra, M. Joseph, B. Radhakrishnan, S. Acharya, and S. Sudarshan. Partial marking for automated grading of sql queries. *VLDBj*, 9(13):1541–1544, Sept. 2016.

[22] G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, N. Polyzotis, and J. S. V. Varman. The QueRIE system for personalized query recommendations. *IEEE Data Eng. Bull.*, 2011.

[23] R. Chinchani, A. Iyer, H. Q. Ngo, and S. Upadhyaya. Towards a theory of insider threat assessment. In *DSN*, 2005.

[24] C. Y. Chung, M. Gertz, and K. Levitt. Demids: A misuse detection system for database systems. In *Integrity and Internal Control in Information Systems*, pages 159–178. Springer, 2000.

[25] Cisco Systems, Inc. 2017 Annual Cybersecurity Report. January 2017.

[26] D. Costa, M. Collins, S. Perl, M. Albrethsen, G. Silowash, and D. Spooner. An ontology for insider threat indicators: Development and application. In *Proc. of the 9th Conference on Semantic Technology for Intelligence, Defense, and Security (STIDS 2014), Fairfax, VA, USA*, pages 48–53. CEUR Workshop Proceedings, November 2014.

[27] E. Costante, S. Vavilis, S. Etalle, J. den Hartog, M. Petković, and N. Zannone. Database anomalous activities detection and quantification. In *Security and Cryptography (SECRYPT), 2013 International Conference on*, pages 1–6. IEEE, 2013.

[28] E. Costante, S. Vavilis, S. Etalle, J. den Hartog, M. Petković, and N. Zannone. Database anomalous activities detection and quantification. In *SECRYPT*, 2013.

[29] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on Android. In *Information Security: Proc. of the 13th International Conference (ISC 2010), Boca Raton, FL, USA*, volume 6531, pages 346–360. Springer, October 2010.

[30] L. Ding, L. Zhou, T. Finin, and A. Joshi. How the semantic web is being used: An analysis of FOAF documents. In *Proc. of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05), Big Island, HI, USA*, pages 113–122. IEEE Computer Society, January 2005.

[31] F. Distel and B. Sertkaya. On the complexity of enumerating pseudo-intents. *Discrete Applied Mathematics*, 159(6):450–466, 2011.

[32] C. Dwork. *ICALP 2006, Proceedings, Part II*, chapter Differential Privacy. Springer, 2006.

[33] G. Falquet, C. Métral, J. Teller, and C. Tweed. *Ontologies in Urban Development Projects*, volume 1 of *Advanced Information and Knowledge Processing*. Springer-Verlag London Limited, 2011.

[34] D. Ferraiolo and J. Barkley. Specifying and managing role-based access control within a corporate intranet. In *ACM RBAC*, 1997.

[35] M. Gafny, A. Shabtai, L. Rokach, and Y. Elovici. Detecting data misuse by applying context-based data linkage. In *Proceedings of the 2010 ACM Workshop on Insider Threats*, Insider Threats '10, pages 3–12, New York, NY, USA, 2010. ACM.

[36] W. Gatterbauer. Databases will visualize queries too. *pVLDB*, 2011.

[37] G. Gavai, K. Sricharan, D. Gunning, J. Hanley, M. Singhal, and R. Rolleston. Supervised and unsupervised methods to detect insider threat from enterprise social and online activity data. *JOWUA*, 2015.

[38] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for OLAP discovery driven analysis. In *ACM DOLAP*, 2009.

[39] J. Glasser and B. Lindauer. Bridging the gap: A pragmatic approach to generating insider threat data. In *2013 IEEE S&P Workshops*, 2013.

[40] J. Golbeck and M. Rothstein. Linking social networks on the web with FOAF: A semantic web case study. In *Proc. of the 23rd National Conference on Artificial Intelligence (AAAI'08), Chicago, IL, USA*. AAAI Press, July 2008.

[41] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *pVLDB*, 1995.

[42] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *DMKD*, 2004.

[43] H. Hedden. *The Accidental Taxonomist*. Information Today, Inc., Medford, New Jersey, 2010. ISBN: 978-1-57387-397-0.

[44] S. Homer and A. L. Selman. *Computability and complexity theory*. Springer Science & Business Media, 2011.

[45] I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *Proc. of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR2006), Lake District, UK*, pages 57–67. AAAI Press, June 2006.

[46] J. Hunker and C. W. Probst. Insiders and insider threats – an overview of definitions and mitigation techniques. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2(1):4–27, March 2011.

[47] C. L. Huth, D. W. Chadwick, W. R. Claycomb, and I. You. Guest editorial: A brief overview of data leakage and insider threats. *Information Systems Frontiers*, 15(1):1–4, March.

[48] A. Kamra, E. Terzi, and E. Bertino. Detecting anomalous access patterns in relational databases. *VLDBJ*, 2007.

[49] N. Karapanos, A. Filios, R. A. Popa, and S. Capkun. Verena: End-to-end integrity protection for web applications. In *IEEE Security and Privacy*, 2016.

[50] O. Kennedy, J. Ajay, G. Challen, and L. Ziarek. Pocket Data: The need for TPC-MOBILE. In *TPC-TC*, 2015.

[51] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. SnipSuggest: context-aware autocompletion for SQL. *pVLDB*, 2010.

[52] P. Kolaitis. Relational databases, logic, and complexity. *Sino-European Winter School in Logic, Language and Computation, Guangzhou, China*, 2010.

[53] M. Krotzsch, F. Simancik, and I. Horrocks. A description logic primer. *arXiv preprint arXiv:1201.4089*, 2012.

[54] G. Kul, D. Luong, T. Xie, P. Coonan, V. Chandola, O. Kennedy, and S. Upadhyaya. Ettu: Analyzing query intents in corporate databases. In *WWW Companion*, 2016.

[55] G. Kul and S. Upadhyaya. A preliminary cyber ontology for insider threats in the financial sector. In *Proc. of the 7th ACM CCS International Workshop on Managing Insider Security Threats (MIST'15), Denver, CO, USA*, pages 75–78. ACM, October 2015.

[56] G. Kul and S. Upadhyaya. Towards a cyber ontology for insider threats in the financial sector. *JOWUA*, 2015.

[57] G. Kul, S. Upadhyaya, and A. Hughes. Complexity of insider attacks to databases. In *Proceedings of the 9th ACM CCS International Workshop on Managing Insider Security Threats*, 2017.

[58] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[59] V. C. S. Lee, J. A. Stankovic, and S. H. Son. Intrusion detection in real-time database systems via time signatures. In *Proceedings Sixth IEEE Real-Time Technology and Applications Symposium. RTAS 2000*, pages 124–133, 2000.

[60] J. Lehmann and J. Voelker. *Perspectives on Ontology Learning*, chapter An introduction to ontology learning. IOS Press, 2014.

[61] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *pVLDB*, 2014.

[62] F. Maggi, W. Robertson, C. Kruegel, and G. Vigna. Protecting a moving target: Addressing web application concept drift. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 21–40, Berlin, Heidelberg, 2009. Springer-Verlag.

[63] V. H. Makiyama, M. J. Raddick, and R. D. Santos. Text mining applied to SQL queries: A case study for the SDSS SkyServer. In *SIMBig*, 2015.

[64] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. Upadhyaya. A data-centric approach to insider attack detection in database systems. In *RAID*, 2010.

[65] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. Upadhyaya. A data-centric approach to insider attack detection in database systems. In *Proc. of the 13th International Conference on Recent Advances in Intrusion Detection (RAID'10), Ottawa, Ontario, Canada*, pages 382–401. Springer-Verlag, September 2010.

[66] S. Mathew, S. Upadhyaya, D. Ha, and H. Q. Ngo. Insider abuse comprehension through capability acquisition graphs. In *Proc. of the 11th International Conference on Information Fusion (FUSION 2008), Cologne, Germany*, pages 1–8. IEEE, June 2008.

[67] R. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *DSN*, 2002.

[68] A. S. McGough, B. Arief, C. Gamble, D. Wall, J. Brennan, J. Fitzgerald, A. van Moorsel, S. Alwis, G. Theodoropoulos, and E. Ruck-Keene. Ben-ware: Identifying anomalous human behaviour in heterogeneous systems using beneficial intelligent software. *JOWUA*, 2015.

[69] R. Mogull. Understanding and selecting a database activity monitoring solution. *Securosis, L.L.C., Sans Institute*, 2013.

[70] D. Mundie, S. Perl, and C. Huth. Toward an ontology for insider threat research: Varieties of insider threat definitions. In *Proc. of the 3rd Workshop on Socio-Technical*

Aspects in Security and Trust (STAST2013), New Orleans, LA, USA, pages 26–36. IEEE, June 2013.

[71] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM CCS*, 2015.

[72] H. V. Nguyen, K. Böhm, F. Becker, B. Goldman, G. Hinkel, and E. Müller. Identifying user interests within the data space-a case study with skyserver. In *EDBT*, 2015.

[73] N. Nguyen, P. Reiher, and G. H. Kuenning. Detecting insider threats by monitoring system call activity. In *IEEE IAW*, 2003.

[74] I. Niles and A. Pease. Towards a standard upper ontology. In *Proc. of the 2nd International Conference on Formal Ontology in Information Systems (FOIS '01), Ogunquit, ME, USA*, pages 2–9. ACM, October 2001.

[75] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. `http://protege.stanford.edu/publications/ontology_development/ontology101.pdf`. Accessed: 2015-06-15.

[76] S. Parkin, A. Fielder, and A. Ashby. Pragmatic security: Modelling it security management responsibilities for sme archetypes. In *ACM CCS MIST*, 2016.

[77] A. Pease, I. Niles, and J. Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*. AAAI Press, 2002.

[78] Ponemon Institute. 2015 Cost of Cyber Crime Study: Global. October 2015.

[79] Ponemon Institute. 2016 Cost of Cyber Crime Study & the Risk of Business Innovation. October 2016.

[80] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[81] S. Pramanik, V. Sankaranarayanan, and S. Upadhyaya. Security policies to mitigate insider threat in the document control domain. In *2004. 20th Annual Computer Security Applications Conference*, pages 304–313. IEEE, 2004.

[82] PwC. The Global State of Information Security Survey 2016. October 2015.

[83] M. B. Salem, S. Hershkop, and S. J. Stolfo. A survey of insider attack detection research. In *Insider Attack and Cyber Security*, pages 69–90. Springer, 2008.

[84] M. B. Salem and S. J. Stolfo. Modeling user search behavior for masquerade detection. In *Proc. of the 14th International Conference on Recent Advances in Intrusion Detection (RAID'11), Menlo Park, CA, USA*, pages 181–200. Springer-Verlag, 2011.

[85] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, (2):38–47, 1996.

[86] R. J. Santos, J. Bernardino, and M. Vieira. Approaches and challenges in database intrusion detection. *SIGMOD Rec.*, 43(3):36–47, Dec. 2014.

[87] A. M. Sanzgiri and S. Upadhyaya. Feasibility of attacks: What is possible in the real world–a framework for threat modeling. In *SAM*, 2011.

[88] C. Sapia. On modeling and predicting query behavior in OLAP systems. In *DMDW*, 1999.

[89] C. Sapia. Promise: Predicting query behavior to enable predictive caching strategies for OLAP systems. In *DaWaK*, 2000.

[90] A. Sarkar, S. Kohler, S. Riddle, B. Ludaescher, and M. Bishop. Insider attack identification and prevention using a declarative approach. In *Proc. of the 2014 IEEE Security and Privacy Workshops (SPW '14), San Jose, CA, USA*, pages 265–276. IEEE, May 2014.

[91] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[92] P. Seshadri, H. Pirahesh, and T. Y. C. Leung. Complex query decorrelation. In *IEEE ICDE*, 1996.

[93] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 2011.

[94] Software Engineering Institute. Analytic Approaches to Detect Insider Threats. December 2015.

[95] W. Stallings. *Cryptography and network security: principles and practices, 7th Edition*. Pearson, 2017.

[96] K. Stefanidis, M. Drosou, and E. Pitoura. "You May Also Like" results in relational databases. In *ACM DOLAP*, 2009.

[97] Y. Sun, H. Xu, E. Bertino, and C. Sun. A data-driven evaluation for insider threats. *Data Science and Engineering*, 1(2):73–85, 2016.

[98] J. Szefer, P. Jamkhedkar, D. Perez-Botero, and R. B. Lee. Cyber defenses for physical attacks and insider threats in cloud computing. In *Proc. of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '14), Kyoto, Japan*, pages 519–524. ACM, June 2014.

[99] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the VLDB Endowment*, volume 6, pages 289–300. VLDB Endowment, 2013.

[100] E. Vanderlinden. Finance ontology documentation. `http://fadyart.com/ontologies/documentation/finance/index.html`, June 2015. Accessed: 2015-06-15.

[101] S. Vavilis, A. Egner, M. Petković, and N. Zannone. An anomaly analysis framework for database systems. *Comput. Secur.*, 53(C):156–173, Sept. 2015.

[102] S. Wang, D. Agrawal, and A. El Abbadi. Hengha: Data harvesting detection on hidden databases. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop*, CCSW '10, pages 59–64, New York, NY, USA, 2010. ACM.

[103] S. Wang, D. Agrawal, and A. El Abbadi. Hengha: Data harvesting detection on hidden databases. In *ACM CCSW*, 2010.

[104] X. Yang, C. M. Procopiuc, and D. Srivastava. Recommending join queries via query log analysis. In *IEEE ICDE*, 2009.

[105] M. J. Zaki and W. Meira Jr. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[106] Y. Zhang, J. K. Daniel Genkin and, D. Papadopoulos, and C. Papamanthou. vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In *IEEE Security and Privacy*, 2017.

[107] Y. Zhang, J. Katz, and C. Papamanthou. Integridb: Verifiable sql for outsourced databases. In *ACM CCS*, 2015.