Data Level Inference Detection in Database Systems*

Raymond W. Yip and Karl N. Levitt
Department of Computer Science
University of California, Davis
One Shields Avenue, Davis CA 95616

Abstract

Existing work on inference detection for database systems mainly employ functional dependencies in the database schema to detect inferences. It has been noticed that analyzing the data stored in the database may help to detect more inferences. In this paper, we describe our effort in developing a data level inference detection system. We have identified five inference rules that a user can use to perform inferences. They are 'subsume', 'unique characteristic', 'overlapping', 'complementary', and 'functional dependency' inference rules. The existence of these inference rules confirms the inadequacy of detecting inferences using just functional dependencies. The rules can be applied any number of times and in any order. These inference rules are sound. They are not necessarily complete, although we have no example that demonstrates incompleteness. We employ a rule based approach so that future inference rules can be incorporated into the detection system. We have developed a prototype of the inference detection system using Perl on a Sun SPARC 20 workstation. The preliminary results show that on average it takes seconds to process a query for a database with thousands of records. Thus, our approach to inference detection is best performed off-line, and would be most useful to detect subtle inference attacks.

1. Introduction

Inference is a method to subvert access control in database systems. An inference occurs when a user is able to infer some data without directly accessing them. In multilevel database systems, early work on inference detection used a graph to represent the functional dependencies among the attributes in the database schema. An inference occurs when there are two or more paths among the attributes, and the paths are labeled at different classification levels [6, 2, 12]. The inference path is eliminated by upgrading some attributes along the path [15, 13]. Lunt [9] points out that some inference problems can be avoided by redesigning the database schema, and classifying the attributes properly. However, redesigning the database schema results in data duplication which leads to update anomalies. It also requires modifications to the existing application programs. There is also work on incorporating external knowledge into the inference detection systems [18, 7, 16, 17, 3]. More recently, researchers suggest using data of the database to generate a richer set of functional dependencies for inference detection. Hinke *et al.* use cardinality associations to discover potential inference paths [8]. Hale *et al.* incorporate imprecise and fuzzy database relations into their inference detection system [5]. However, existing efforts still simply employ functional dependencies to detect inferences. As noted by SRI researchers, monitoring user activities may lead to detecting more inferences [14]. By *data level inference detection*, we mean the system detects inferences by considering the data in the database, as opposed to the database schema only.

Inferences can also occur in discretionary access control systems where users are explicitly granted access rights to access data (as in System R). It is not an obvious task to grant users the exact amount of access rights they need. In some cases, users are simply granted more access rights than they need in order not to hinder their work. To ensure the users do not misuse the database, we need to monitor their accesses.

A simple way to monitor user accesses is to examine each user query, and reject any query that accesses sensitive data. However, it is possible for a user to use a series of unsuspicious queries to infer data in the database. Motro *et al.* address a similar problem, but

 Published in *Proceedings, 1998 IEEE Computer Security Foundations Workshop*, Rockport, Massachusetts, June, 1998.

their work focuses on detecting aggregation instead of inference attacks [11]. In the statistical database security community, various techniques have been proposed to protect individual records, for example, query-set-size control, cell suppression, and data perturbation [1]. However, these techniques are not suitable for detecting inferences using general purpose queries.

In this paper, we describe our effort in developing a data level inference detection system. It is a static inference detection system where inferences are performed with respect to a snapshot of the database. We have identified five inference rules that users can use to infer data: 'subsume', 'unique characteristic', 'overlapping', 'complementary', and 'functional dependency' inference rules. Users can apply these rules any number of times, and in any order to infer data. These inference rules are sound but not necessarily complete. Although we have no example that demonstrates incompleteness, more research effort is needed to determine if they are complete. We employ a rule based approach so that when a new inference rule is discovered, it can be incorporated into the inference detection system. The existence of these inference rules confirms the inadequacy of functional-dependency based inference detection schemes. We have developed a prototype of the inference detection system to study its performance. The preliminary results show that on average the system takes a few seconds to process a query that returns hundreds of records from a database of ten thousand records. Thus, our approach to inference detection is best performed off-line, and would be most useful to detect subtle inference attacks.

This paper is organized as follows. In Section 2, we provide the intuition behind the five inference rules. In Section 3, we introduce the notations used in this paper. In Section 4, we present the five inference rules. In Section 5, we describe our implementation and the preliminary results. We give our conclusions in Section 6.

2. Overview of the Inference Rules

In this section, we provide the intuition behind the five inference rules. The goal of our inference detection system is to detect if a user can indirectly access data using two or more queries. In particular, the system determines if the user can infer the return tuples from different queries corresponding to the same tuple in the database.

The result of each user query is a set of return tuples [1]. The user cannot identify each return tuple unless the primary key of the tuple is also returned. However, a certain group of attribute values of a tuple may uniquely identify the tuple. The unique identification rule handles this situation. Another way to identify a return tuple is to compare it with other return tuples that have already been identified.

There are two possible relationships between two sets of return tuples. One possibility is that for each return tuple t_1 of a query, there is a return tuple t_2 of the other query, such that t_1 and t_2 correspond to the same tuple in the database. The subsume inference rule handles this case. Another possibility is that only some return tuples of a query correspond to some return tuples of another query. The overlapping inference rule identifies the corresponding return tuples that are common to both queries. The complementary inference rule identifies the corresponding return tuples by taking the "difference" between two sets of return tuples. The functional dependency inference rule is introduced to simulate the schema level inference detection scheme.

Once the corresponding return tuples between two queries are identified, the user can generate *inferred queries*. The user knows the return tuples of an inferred query without directly issuing it to the database. For example, the user can infer a new query with returns tuples "common" to both queries, or a new query that returns tuples from one query but not from another query. The user can also combine several queries into a single query. We will discuss the effect of applying inference rules to unions of queries. Essentially, the five inference rules cover the set intersection, difference and union relationships between two sets of return tuples.

When the user issues a query, the inference detection system compares it with previously issued queries and inferred queries, and applies inference rules when appropriate. An occurrence of inference will result in either the modifications of the existing queries (for example, combining two corresponding return tuples), or the generation of new inferred queries. These may trigger further applications of the inference rules. Hence, the inference rules are applied repeatedly until there is no new inference occurs. This is a terminating process as the number of inferences that can occur is bounded by the size of the database. When two users are suspected of cooperating in performing inference, we can run the inference detection system against their combined set of queries.

[1] Unless otherwise stated, a set of return tuples is indeed a multiset of return tuples. That is, duplicated return tuples are not removed

3. Preliminaries and Notation

We consider inference detection in a relational database with a single table. A database with multiple tables can be transformed into a universal relation [10]. We assume that the only way the user can learn about the data in the database is by issuing queries to it. That is the user does not rely on real-world knowledge to perform any inference. Such knowledge might be added to the database as 'catalytic relation' as suggested in [6].

A_i denotes an attribute in the table, and a_i denotes an attribute value from the domain of A_i. $t[A_i]$ denotes the attribute value of a single tuple t over the attribute A_i. A query is represented by a 2-tuple: *(attribute-set, selection-criterion)*, where *attribute-set* is the set of attributes projected by the query, and *selection-criterion* is the logical expression that is satisfied by each return tuple of the query. No aggregation function (for example, maximum and average) is allowed in the attribute-set. In general, Q_i refers to the query $\{AS_i; SC_i\}$. $|Q_i|$ denotes the number of return tuples of Q_i. $\{Q_i\}$ denotes the set of return tuples of Q_i. For each query Q_i, AS_i is expanded with an attribute A_i when '$A_i = a_i$' appears in SCi as a conjunct. A query Q is a *'partial query'* if the user can determine $|Q|$, but not all return tuples of Q. '$\cap$', '$\cup$', and '$\setminus$' stand for the set intersection, union, and difference operation respectively.

Now, we introduce several notions that are used throughout this paper.

Definition 1 *A tuple t over a set of attributes AS 'satisfies' a logical expression E if E is evaluated to true when each occurrence of A_i in E is instantiated with $t[A_i]$, for all A_i in AS. t 'contradicts' with E if E is evaluated to false.*

For example, the tuple *(35, 60K)* that is projected over *(Age, Salary)* satisfies $E =$ *(Age* $> 30 \wedge$ *Salary* $< 70K)$; while the tuple *(25, 50K)* projected over the same set of attributes contradicts with E. The tuple *(45K, Manager)* projected over *(Salary, Job)* neither satisfies nor contradicts E. This is because after the instantiation, E becomes (Salary $<$ 70K) whose truth value is undetermined.

Definition 2 *Given two queries, Q_1 and Q_2, we say that Q_1 is 'subsumed' by Q_2, denoted as $Q_1 \sqsubset Q_2$, iff*

1. *$SC_1 \rightarrow SC_2$; or*
2. *for each tuple t_1 in $\{Q_1\}$, t_1 satisfies SC_2.*

where $\rightarrow$ is the logical implication. '$\sqsubset$' is a reflexive, anti-symmetric, and transitive relation. A return tuple t_1 *'relates'* to another return tuple t_2 if the two tuples are selected from the same tuple in the database. Hence, $Q_1 \sqsubset Q_2$ implies that for each return tuple t_1 of Q_1, there is a return tuple t_2 of Q_2, such that t_1 relates to t_2.

When evaluating a logical implication, we need to consider the integrity constraints that hold in the database. Consider the following implication,

$(\mathsf{age} > 18 \wedge \mathsf{age} < 35) \rightarrow (\mathsf{age} > 20 \wedge \mathsf{age} < 50)$,

which is false. Suppose the youngest person in the database is 22 years old. By adding this constraint to both sides of the implication, it becomes,

$((\mathsf{age} > 18 \wedge \mathsf{age} < 35) \wedge (\mathsf{age} \geq 22)) \rightarrow$
$((\mathsf{age} > 20 \wedge \mathsf{age} < 50) \wedge (\mathsf{age} \geq 22)) \equiv$
$(\mathsf{age} \geq 22 \wedge \mathsf{age} < 35) \rightarrow$
$(\mathsf{age} \geq 22 \wedge \mathsf{age} < 50)$,

which is true.

Now, we introduce the notion of 'indistinguishable'.

Definition 3 *A return tuple $t1$ of $Q1$ is 'indistinguishable' from a return tuple $t2$ of $Q2$ iff*

1. *for all A_i in $(AS_1 \cap AS_2)$, $t_1[A_i] = t_2[A_i]$;*
2. *t_1 does not contradict with SC_2; and*
3. *t_2 does not contradict with SC_1.*

t_1 is 'distinguishable' from t_2 if t_1 is not indistinguishable from t_2.

Intuitively, t_1 is indistinguishable from t_2 if it is not possible to conclude that t_1 and t_2 are selected from two different tuples in the database. Two tuples that relate to each other are indistinguishable from each other, while two tuples that are indistinguishable from each other does not imply that they relate to each other.

Our system only detects inferences of the "right instances" of the data. Consider the following table,

Name	Job	Age	Salary
John	Engineer	29	60K
Paul	Engineer	31	60K

Suppose the user knows that John is an engineer, and that there is an engineer who is 31 years old and earns 60K. A naive user may conclude that John earns 60K, assuming that John's age is 31. Although the user correctly infers the salary of John, this is not the right instance of the salary of John. In fact, when the user learns that John is indeed 29, the user will revoke this inference. In our model, a skeptical user will not make such hasty inferences.

Name	Job	Age	Salary	Department	Office
Alice	Manager	35	60K	Marketing	2nd Floor
Bob	Secretary	35	45K	Marketing	2nd Floor
Charles	Secretary	40	40K	Production	1st Floor
Denise	Manager	45	65K	Sales	2nd Floor

Figure 1. Sample database.

4. Inference Rules

In this section, we describe the five inference rules. We illustrate the inference rules using the sample database as shown in Figure 1. The user is allowed to access all data in the database. However, it is suspicious if the user can infer the salaries of employees. We assume the the security policy is to determine if the user infer the associations between *'Name'* and *'Salary'*. In general, the policy can specify detecting inferences of any association among the attributes. Unless otherwise stated, all queries appear in the inference rules are not partial queries.

4.1. Subsume Inference

In this section, we describe the use of the '$\sqsubset$' relations to perform inferences.

Inference Rule 1 (Subsume) *Given two queries Q_1 and Q_2, such that $Q_1 \sqsubset Q_2$.*

SI1 *If there is an attribute A in $(AS_2 \setminus AS_1)$, such that all return tuples of Q_2 take the same attribute value a over A, then for each return tuple t_1 of Q_1, $t_1[A] = a$. Q_1 may be a partial query.*

SI2 *If there is a return tuple t_1 of Q_1 that is indistinguishable from one and only one return tuple t_2 of Q_2, then t_1 relates to t_2. Q_1 may be a partial query.*

SI3 *Let S be the set of return tuples of Q_2 that are distinguishable from the return tuples of Q_1. If $|S| = (|Q_2| - |Q_1|)$, then two inferred queries are generated: $(AS_2; SC_2 \wedge \neg SC_1)$ with S as the set of return tuples, and $(AS_2; SC_2 \wedge SC_1)$ with $(\{Q_2\} \setminus S)$ as the set of return tuples. If $|S| < (|Q_2| - |Q_1|)$, then an inferred partial query is generated: $(AS_2; SC_2 \wedge \neg SC_1)$ with S as the partial set of return tuples.*

$Q_1 \sqsubset Q_2$ implies that for each return tuple t_1 of Q_1, there is a return tuple t_2 of Q_2, such that t_1 relates to t_2. *SI1* says that when all return tuples of Q_2 share a common attribute value, say a, over an attribute A, the user can infer that each return tuple of Q_1 also take the attribute value a over the attribute A. For example, consider the following two queries:

Q_1 = (Age; Name = 'Alice'), and

Q_2 = (Department; Age < 40).

Q_1 returns a single tuple *(35)* which says that Alice is 35 years old. Q_2 returns two tuples *('Marketing')* and *('Marketing')* which shows that all employees at the age less than 40 work in the Marketing department. By *SI1*, Alice works in the Marketing department.

SI2 says that if t_2 is the only return tuple of Q_2 that is indistinguishable from a return tuple t_1 of Q_1, then t_1 relates to t_2. Consider the following two queries:

Q_3 = (Age; Name = 'Charles'), and

Q_4 = (Age, Salary; Age $\geq$ 40).

Q_3 returns a single tuple t_3 = *(40)* which says that Charles is 40 years old. Q_4 returns two tuples *(40, 40K)* and *(45, 65K)* which says that there are only two employees who are at the age greater than or equal to 40. As $Q_3 \sqsubset Q_4$ (since *(40)* satisfies SC_4) and *(40, 40K)* is the only return tuple of Q_4 that is indistinguishable from t_3, by *SI2*, Charles earns 40K.

SI3 says that if a user identifies all the return tuples of Q_2 that relate to the return tuples of Q_1, then the user can infer these two queries: $(AS_2; SC_1 \wedge SC_2)$ which includes return tuples of Q_2 that relate to the return tuples of $Q1$, and $(AS_2; SC_2 \wedge \neg SC_1)$ which includes return tuples of Q_2 that do not relate to the return tuples of Q_1. Continue from the above example on Q_3 and Q_4, after the application of *SI2*, we generate the following two inferred queries:

Q_{21} = (Age, Salary; Name = 'Charles' $\wedge$ Age $\geq$ 40), and

Q_{22} = (Age, Salary; Name $\neq$ 'Charles' $\wedge$ Age $\geq$ 40).

Q_{21} returns a single tuple *(40, 40K)*, and Q_{22} returns a single tuple *(45, 65K)*. The two inferred queries together contains more information than Q_2. For example, Q_{22} says that the employee who is at the age of 45 and earns 65K must be someone other than Charles.

4.2. Unique Characteristic Inference

A unique characteristic is defined as follows,

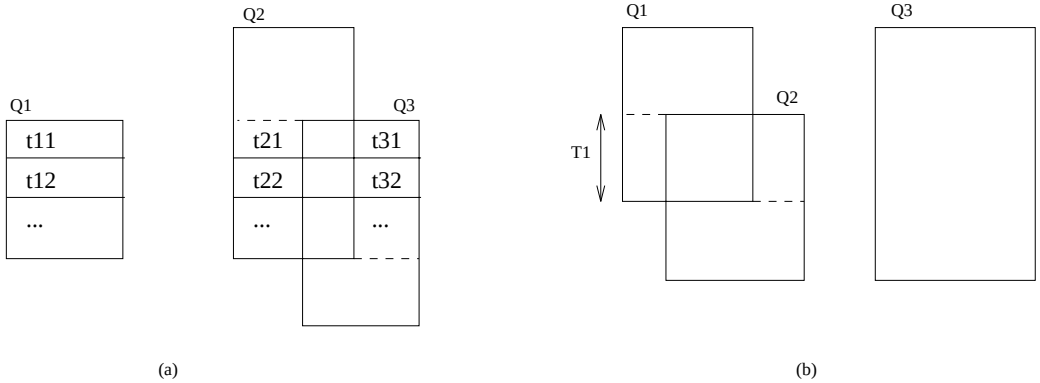

Figure 2. Examples on overlapping inference.

Definition 4 *A logical expression E is a unique characteristic of a tuple t iff t is the only tuple in the database that satisfies E.*

For example, if Alice is the only manager at the age of 35, then *(Job = 'Manager' ∧ Age = 35)* is the unique characteristic of Alice in the database.

Inference Rule 2 (Unique Characteristic) *Given a tuple t_1 with unique characteristic C_1 in a database D, and another tuple t_2 with unique characteristic C_2 in D. If $C_1 \rightarrow C_2$, $C_2 \rightarrow C_1$, or $C_1 \leftrightarrow C_2$ (that is $C_1 \rightarrow C_2$ and $C_2 \rightarrow C_1$), then t_1 relates to t_2 in D.*

For example, the query

(Salary; Job = 'Manager' ∧ Age ≤ 40)

returns a single tuple *(60K)*. This query together with the above unique characteristic of Alice implies Alice earns 60K. Unique characteristic inference is a special case of the subsume inference. Suppose $(AS_1; UC_1)$ returns a single tuple t_1, and $(AS_2; UC_2)$ returns a single tuple t_2. Then, UC_1 is the unique characteristic of t_1, and UC_2 is the unique characteristic of t_2. If $UC_1 \rightarrow UC_2$, $UC_2 \rightarrow UC_1$, or $UC_1 \leftrightarrow UC_2$ holds, then by *SI2*, t_1 relates to t_2.

If all inferred queries are identified, unique characteristics are determined as follows,

1. if Q_i returns all but one tuple t in the database, then the unique characteristic of t is $(\neg SC_i)$.

2. if Q_i and Q_j have only one overlapping return tuple t, then t has the unique characteristic $(SC_i \wedge SC_j)$.

3. if Q_i returns one more tuple t than Qj, then the unique characteristic of t is $(SC_i \wedge \neg SC_j)$.

where both Q_i and Q_j are not partial queries. The determination of the overlapping tuples between two queries is discussed in the Section 4.3.

4.3. Overlapping Inference

In this section, we describe the overlapping inference rule.

Inference Rule 3 (Overlapping) *Given n queries $Q_1, \ldots, Q_n$, where $n \geq 3$.*

OI1 *Let S be the set of return tuples of Q_2 that are indistinguishable from the return tuples of Q_3. If $Q_1 \sqsubset Q_2$, $Q_1 \sqsubset Q_3$, $|S| = |Q_1|$, and t_2 is the only return tuple of Q_2 that is indistinguishable from a return tuple t_3 of Q_3, then t_2 relates to t_3. Q_1 may be a partial query.*

OI2 *Let $QS = \{Q_2, \ldots, Q_n\}$. Suppose for each Q_i in QS, $Q_i \sqsubset Q_1$, and the return tuples of Q_i are indistinguishable from the return tuples of at most one other query in QS. Also, the total number of indistinguishable tuples in all queries in QS is equal to $(2 \times (|Q_2| + \ldots + |Q_n| - |Q_1|))$. For any two queries Qj and Qk in QS, if t_j is the only return tuple of Q_j that is indistinguishable from a return tuple t_k of Q_k, then t_j relates to t_k. Q_1 may be a partial query.*

OI3 *When all relating tuples between Q_i and Q_j are identified, three inferred queries are generated (possibly partial): $(AS_i; SC_i \wedge \neg SC_j)$, $(AS_j; SC_j \wedge \neg SC_i)$, and $(AS_i \cap AS_j; SC_i \wedge SC_j)$.*

Figure 2(a) illustrates *OI1*. Each rectangle represents a set of return tuples of a query. The rectangles are drawn in such a way that return tuples that are selected from the same tuple in the database are aligned

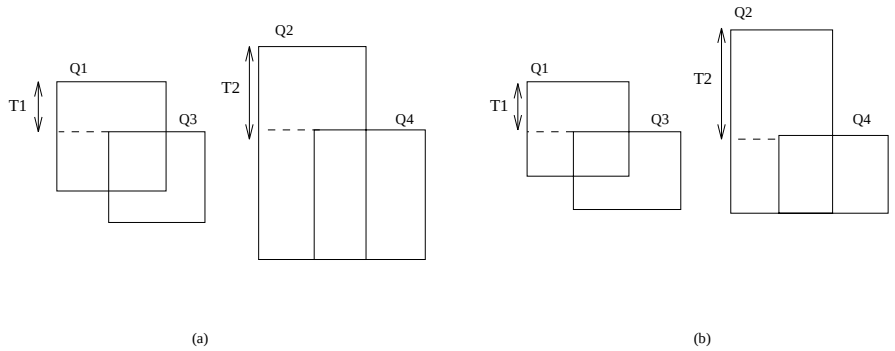

Figure 3. Examples on complementary inference.

horizontally. For example, t_{11}, t_{21}, and t_{31} correspond to the same tuple in the database. Suppose $Q_1 \sqsubset Q_2$ and $Q_1 \sqsubset Q_3$, and the number of indistinguishable tuples between Q_2 and Q_3 equals $|Q_1|$. This implies that for each return tuple t_{11} of Q_1, there is a return tuple t_{21} of Q_2, and a return tuple t_{31} of Q_3, such that t_{11} relates to t_{21}, and t_{11} relates to t_{31}; that is, t_{21} relates to t_{31}. We further illustrate *OI1* by an example. Consider the following three queries,

Q_1 = (Name; Job = 'Manager' $\wedge$ Age = 35),
Q_2 = (Salary; Job = 'Manager'), and
Q_3 = (Salary; Age = 35).

Q_1 returns a single tuple *(Alice)* which says that Alice is the only manager at the age of 35. Q_2 returns two tuples *(60K)* and *(65K)*. Q_1 and Q_2 together implies that the salary of Alice is either 60K or 65K. Q_3 returns two tuples *(60K)* and *(45K)*. Q_1 and Q_3 together implies that the salary of Alice is either 60K or 45K. As $Q_1 \sqsubset Q_2$ and $Q_1 \sqsubset Q_3$, and there is only one return tuple of Q_2 that is indistinguishable from return tuples of Q_3, namely the tuple *(60K)*. Hence, by *OI1*, Alice earns 60K. When Q_1 implies three or more queries, *OI1* is applied to two of them at a time.

Figure 2(b) illustrates *OI2*. Let T_1 be the set of return tuples of Q_1 that relate to return tuples of Q_2. $(|Q_1| + |Q_2| - |T_1|)$ is the number of tuples in both Q_1 and Q_2 that do not relate to one another. When $(|Q_1| + |Q_2| - |T_1|) = |Q_3|$, the user can infer that for each return tuple t_1 of Q_1 that is indistinguishable from a return tuple t_2 of Q_2, t_1 relates to t_2. As the tuples that are indistinguishable from each other appear in exactly two queries, the number of indistinguishable tuples equals $2 \times (|Q_1| + |Q_2| - |Q_3|)$. We further illustrate *OI2* with the following three queries,

Q_1 = (Salary; Department = 'Marketing' $\wedge$ Office = '2nd Floor'),
Q_2 = (Salary; Job = 'Manager' $\wedge$ Office = '2nd Floor'), and
Q_3 = (Name; Office = '2nd Floor').

Q_1 returns two tuples *(60K)* and *(45K)* which says that the two employees who work in the Marketing department on the 2nd floor earn either 60K or 45K. Q_2 returns two tuples *(60K)* and *(65K)* which says that the two managers who work on the 2nd floor earn either 60K or 65K. Q_3 returns three tuples *(Alice)*, *(Bob)*, and *(Denise)* which says that Alice, Bob and Denise all work on the 2nd Floor. We have 1) $Q_1 \sqsubset Q_3$, 2) $Q_2 \sqsubset Q_3$, 3) there is only one return tuple of Q_1 that is indistinguishable from a return tuple of Q_2, namely the tuple {*(60K)*} (that is, the number of indistinguishable tuples in both Q_1 and Q_2 is 2), and 4) $2 \times (|Q_1| + |Q_2| - |Q_3|) = 2 \times (2 + 2 - 3) = 2$. By *OI2*, the tuple *(60K)* of Q_1 relates to the tuple *(60K)* of Q_2. That is, the user can infer that the marketing manager who works on the 2nd floor earns 60K.

4.4. Complementary Inference

Complementary inference rule performs inferences by eliminating tuples that are not relating to one another.

Inference Rule 4 (Complementary Inference) *Given four queries, Q_1, Q_2, Q_3, and Q_4, where $Q_1 \sqsubset Q_2$, and $Q_3 \sqsubset Q_4$. Also, the return tuples of $Q1$ that relate to the return tuples of $Q3$ are identified (for example using the overlapping inference rule), and similarly for those between $Q2$ and $Q4$. If one of the following three conditions holds,*

1. *for each return tuple t_1 of Q_1 that does not relate to any return tuple of Q_3, t_1 is distinguishable from all return tuples of Q_4,*
2. *$Q_4 \sqsubset Q_3$, or*
3. *$|Q_3| = |Q_4|$,*

then $Q'_1 \sqsubset Q'_2$, where $Q'_1 = (AS_1; SC_1 \wedge \neg SC_3)$, and $Q'_2 = (AS_2; SC_2 \wedge \neg SC_4)$.

Figure 3(a) illustrates the case where condition (1) holds. Let T_1 be the set of return tuples of Q_1 that do not relate to any return tuple of Q_3, and T_2 be the set of return tuples of Q_2 that do not relate to any return tuple of Q_4. As $Q_1 \sqsubset Q_2$ and $T_1 \subset \{Q_1\}$, each tuple in T_1 relates to a return tuple of Q_2. Condition (1) says that each tuple in T_1 does not relate to any return tuple of Q_4. Hence, each tuple in T_1 relates to a tuple in T_2. Figure 3(b) illustrates the case where condition (2) or (3) holds. Condition (2) or (3) implies that $Q_3 \sqsubset Q_4$ and $Q_4 \sqsubset Q_3$. By removing from Q_1 and Q_2 the "same" set of return tuples, we have $Q_1' \sqsubset Q_2'$. We further illustrate this by an example. Consider the following four queries,

Q_1 = (Name; Department = 'Marketing'),
Q_2 = (Salary; Department = 'Marketing' $\vee$ Office = '2nd Floor'),
Q_3 = (Name; Job = 'Secretary'), and
Q_4 = (Salary; Job = 'Secretary').

Q_1 returns two tuples *(Alice)* and *(Bob)*. Q_2 returns three tuples *(60K)*, *(45K)*, and *(65K)*. As $Q_1 \sqsubset Q_2$, both Alice and Bob earn either 60K, 45K, or 65K. Q_3 returns two tuples *(Bob)* and *(Charles)*. Q_4 returns two tuples *(45K)* and *(40K)*. As $SC_3 = SC_4$, both Bob and Charles earn either 45K or 40K. Now, we have $Q1 \sqsubset Q2$, $Q_3 \sqsubset Q_4$, and $Q_4 \sqsubset Q_3$, *(Bob)* is the only related tuple between Q_1 and Q_3 (assuming the employee names are unique), *(45K)* is the only related tuple between Q_2 and Q_4 (it is the only indistinguishable tuple between Q_2 and Q_4). By the complementary inference rule, $Q_1' \sqsubset Q_2'$, where

Q_1' = (Name; Department = 'Marketing' $\wedge$ Job $\neq$ 'Secretary')
Q_2' = (Salary; Department = 'Marketing' $\vee$ Office = '2nd Floor' $\wedge$ Job $\neq$ 'Secretary')

Q_1' returns a single tuple *(Alice)*, as it is the tuple returned by Q_1 but not by Q_3. Q_2' returns two tuples *(60K)* and *(65K)*, as they are the tuples returned by Q_2 but not by Q_4. Therefore, the user can infer that Alice earns either 60K or 65K.

4.5. Functional Dependency Inference

The functional dependency inference rule employs the functional dependencies among the attributes to perform inferences. It simulates the uses of functional dependencies in schema level inference detection systems.

Inference Rule 5 (Functional Dependency) *Given that attribute A_1 functional determines attribute A_2, and there exists a tuple t, such that $t[A_1] = a_1$ and $t[A_2] = a_2$. If there is a tuple t_i, such that $t_i[A_1] = a_1$, then $t_i[A_2] = a_2$. The same applies when A_1 or A_2 is a composite attribute (that is, a group of attributes).*

For example, if it is known that the attribute 'Department' functionally determines the attribute 'Office', and in particular the Marketing department is located on the 2nd Floor. Then, whenever a user knows a person who works in the Marketing department, the user knows the office of that person is located on the 2nd Floor. A similar rule exists for multivalue functional dependencies.

4.6. Inference with Union Queries

In this section, we discuss the use of a union of queries in inferences. Consider the following three queries,

Q_1 = (Job; Age $<$ 50 $\wedge$ Age $>$ 40),
Q_2 = (Job; Age $>$ 45 $\wedge$ Age $<$ 60), and
Q_3 = (Job; Age $>$ 30 $\wedge$ Age $\leq$ 45).

since the following implication holds,

(Age $<$ 50 $\wedge$ Age $>$ 40) $\rightarrow$
((Age $>$ 45 $\wedge$ Age $<$ 60) $\vee$ (Age $>$ 30 $\wedge$ Age $\leq$ 45)),

$Q_1 \sqsubset (Q_2 \cup Q_3)$ holds. The inference rules can still be applied by treating $(Q_2 \cup Q_3)$ as a single user query. We call such a union of queries a *'union query'*. In contrast, a user query is called a *'simple query'*. If Q_u is a union query that consists Q_i, ..., and Q_j, then $AS_u = (AS_i \cap \ldots \cap AS_j)$, and $SC_u = (SC_i \vee \ldots \vee SC_j)$. The applications of the unique characteristic and functional dependency inference rules on a union query are the same as their applications on the simple queries of the union query. Hence, we only consider the applications of the subsume, overlapping, and complementary inference rules on union queries.

Consider the applications of the subsume inference rule on union queries. Suppose $(Q_2 \cup Q_3) \sqsubset Q_1$. This implies that ($Q_2 \sqsubset Q_1$ and $Q_3 \sqsubset Q_1$). If the subsume inference rule is applicable due to $(Q_2 \cup Q_3) \sqsubset Q_1$, then it is also applicable due to ($Q_2 \sqsubset Q_1$) and ($Q_3 \sqsubset Q_1$). Hence, we do not need to consider the application of the subsume inference rule when the union query occurs on the left hand side of a '$\sqsubset$' relation. Now, suppose $Q_1 \sqsubset Q_u$, where Q_u is a union query. The application of *SI1* on union queries is the same as when only simply queries are involved. To apply *SI2*, the user must has identified all the overlapping tuples among the simple queries of Q_u that correspond to the return tuples of Q_1. The subsume inference rule can still be applied when the simple queries of Q_u have no common projected attribute.

Consider the applications of the overlapping inference rule on union queries. Firstly, consider the application of *OI1*. If $Q_u \sqsubset Q_1$ is involved, $|Q_u|$ must be

known to the user. If $Q_1 \sqsubset Q_u$ is involved, the user must has identified all the overlapping tuples among the simple queries of Q_u that relate to the return tuples of Q_1. Now, consider the application of *OI2*. If $Q_u \sqsubset Q_1$ is involved, the user must has identified all the overlapping tuples among the simple queries of Q_u that relate to the return tuples of Q_1. If $Q_1 \sqsubset Q_u$ is involved, $|Q_u|$ must be known to the user. In either case, the attribute set of the union query cannot be empty.

To apply the complementary inference rule on union queries, the overlapping tuples of the simple queries in the union query must have been identified. Also the attribute set of the union query cannot be empty.

Finding all the eligible union queries that have the '$\sqsubset$' relations with other queries is an NP-hard problem. This is because for each simple query Q_1, we need to find all union queries Q_u such that $Q_1 \sqsubset Q_u$ holds. This becomes the problem of finding a set of simple queries that together returns a set of tuples that covers another set of return tuples. If we can solve this problem, we can also solve the set-covering problem which is known to be an NP-complete problem [4]. We have developed a prototype to study the performance of the inference detection system in practice. It is discussed in Section 5.

5. Implementation and Preliminary Results

We have developed a prototype of the inference detection system in about 4,000 lines of Perl code. We have implemented the subsume, unique characteristic, overlapping, and complementary inference rules. We run our experiments with randomly generated tables and user queries. Each table has N_{attr} number of attributes, and N_{rec_num} number of records. The primary key of the table is a single attribute. All attributes are of integer types. Each attribute value in the table is uniformly distributed between 0 and $(N_{data\_dist} \times N_{rec\_num})$, where $0 < N_{data\_dist} \leq 1$. We also randomly generate N_{query_num} number of user queries. Each query projects N_{proj} number of attributes from the table. The selection criterion of each query is a conjunction of N_{cond} number of conjuncts. Each conjunct is of the form 'A_i *op* a_i', where A_i is an attribute from the table, *op* is one of the relational operations ($>$, $\geq$, $\leq$, $<$, and $=$), and a_i is an attribute value. We only consider queries with the number of return tuples falls between 1 and $(N_{set\_size} \times N_{rec\_num})$, where $0 < N_{set\_size} \leq 1$. We approximate the evaluation of a logical implication $C_i \rightarrow C_j$ by checking if the tuples selected by C_i is also selected by C_j, and that the set of attributes appears in C_j is a subset of those appear in C_i.

The preliminary results of running the inference detection system on a Sun SPARC 20 workstation are shown in Table 1–Table 4. In each experiment, we run the inference detection system against 500 user queries. We have collected data about 1) the average number of seconds used to process one query; 2) the number of inferred queries generated; 3) the number of times the inference rules are applied; and 4) the ratio between the number of attribute values of those individual records that have been identified by the user (either by directly accessing them using a query or by inferences) and the total number of attribute values in the database. The ratio is denoted as "% of DB revealed". For example, consider the two queries Q_1 and Q_2 in Section 4.1. These two queries together reveals that Alice is 35 years old, and she works in the Marketing department. Hence, the number of attribute values revealed to the user is 3 (namely Alice's name, Alice's age, and Alice's department). Note that although Q_2 returns two tuples, the user cannot determine whom these two tuples belong to; hence, they are not included as the attribute values that are revealed to the user. The total number of attribute values in the sample database is 24 (there are four records, each with 6 attribute values). Hence, the '% of DB revealed' by Q_1 and Q_2 to the sample database is (3 / 24) $\times$ 100%, or 12.5%.

Table 1 shows the results for N_{rec_num} = 1000, N_{set_size} = 10%, N_{proj} = 4, N_{cond} = 3, N_{query_num} = 500, N_{data_dist} takes the values of 33%, 66%, and 100%, and N_{attr} takes the values of 50, 70, and 90. The number of tuples returned by each query is about 30. It shows that the system performs better as N_{attr} increases. This is because the larger the number of attributes in the table, the lesser the chance that the '$\sqsubset$' relations hold among queries. Also, the system performs better when N_{data_dist} decreases. The lower the distribution of the data, the more duplication of the data values, the lesser the chance a return tuple will be distinguishable from others, and hence the smaller number of occurrences of inferences. Table 2 shows similar results for N_{rec_num} equals 10,000. The number of tuples returned by each query is about 200.

Table 3 shows the results for N_{rec_num} = 1000, N_{data_dist} = 50%, N_{attr} = 80, N_{proj} = 4, N_{cond} = 3, and N_{query_num} = 500, and N_{set_size} takes the values of 10%, 20%, 30%, 40%, and 50%. It shows that the system performs better when N_{set_size} decreases. This is because the more the number of records returned by the queries, the more the number of occurrences of inferences, and also the more the number of inferred queries being generated.

N_{data_dist} (%)	N_{attr}	average query processing time (sec)	number of inferred queries	number of inferences	% of DB inferred
33	50	1.990	40	3	6.94
33	70	1.622	14	0	3.42
33	90	1.532	7	0	3.06
66	50	2.208	49	5	8.30
66	70	1.632	18	0	3.45
66	90	1.608	7	0	3.74
100	50	2.386	50	11	8.79
100	70	1.642	18	0	3.78
100	90	1.650	9	0	3.95

Table 1. N_{rec_num} **= 1000,** N_{set_size} **= 10%,** N_{proj} **= 4,** N_{cond} **= 3,** N_{query_num} **= 500.**

N_{data_dist} (%)	N_{attr}	average query processing time (sec)	number of inferred queries	number of inferences	% of DB inferred
33	50	3.876	53	14	6.12
33	70	4.534	27	9	5.67
33	90	2.484	13	0	1.82
66	50	4.948	65	19	7.01
66	70	5.534	35	9	6.13
66	90	2.848	19	0	2.23
100	50	6.160	87	99	7.45
100	70	6.618	45	39	6.34
100	90	3.002	19	0	2.29

Table 2. N_{rec_num} **= 10000,** N_{set_size} **= 10%,** $Nproj$ **= 4,** $Ncond$ **= 3.**

N_{set_size} (%)	average query processing time (sec)	number of inferred queries	number of inferences	% of DB inferred
10	1.700	34	0	1.85
20	1.890	23	0	5.77
30	1.956	20	3	5.76
40	2.152	22	3	8.08
50	2.168	16	3	8.03

Table 3. N_{rec_num} **= 1000,** N_{data_dist} **= 50%,** N_{attr} **= 80,** N_{proj} **= 4,** N_{cond} **= 3,** N_{query_num} **= 500.**

N_{proj}	N_{cond}	average query processing time (sec)	number of inferred queries	number of inferences	% of DB inferred
4	3	1.700	34	0	1.85
4	4	1.400	3	0	1.93
4	5	1.452	3	0	2.76
4	6	1.432	1	0	2.31
3	3	1.346	5	0	2.22
5	3	1.800	28	0	4.29
6	3	2.556	68	0	7.30

Table 4. N_{rec_num} **= 1000,** N_{data_dist} **= 50%,** N_{attr} **= 80,** N_{set_size} **= 10%.**

Table 4 shows the results for $N_{rec\_num} = 1000$, $N_{data\_dist} = 50\%$, $N_{attr} = 80$, $N_{set\_size} = 10\%$, N_{proj} takes the values of 3, 4, 5, and 6 while N_{cond} is kept constant at 3, and N_{cond} takes the values of 3, 4, 5, and 6 while N_{proj} is kept constant at 4. It shows that the system performs better when N_{proj} decreases. This is because the more the number of attributes projected by the queries, the more overlapping among the queries, and hence the more number of inferences can occur. Also, the system performs better when N_{cond} increases. This is because with a larger number of conjuncts in the selection criteria of the queries, there is lesser chance that the '$\sqsubset$' relations hold among the queries, and hence the smaller number of occurrences of inferences.

In general, we expect to see the inference detection system performs better with the larger number of attributes in the table, the more duplication of attribute values in the database, the smaller number of records returned by queries, the smaller number of attributes projected by the queries, and the larger number of conjuncts in the selection criteria of the queries.

6. Conclusions

In this paper, we describe our effort in developing a data level inference detection system. We have identified five inference rules: subsume, unique characteristic, overlapping, complementary, and functional dependency inference rules. These rules are sound but they are not necessarily complete. The existence of these inference rules shows that simply using functional dependencies to detect inferences is inadequate. We have developed a prototype of the inference detection system using Perl on a Sun SPARC 20 workstation. The preliminary results show that the system on average takes seconds to process a query for a database of thousands of records.

Although in theory detecting inferences at data level is an NP-hard problem, in practice, there are cases where the use of such approach is practical. In particular, this is the case when there is a limited amount of overlapping among the return tuples of the queries. We can further improve the system performance using distributed computing techniques. For example, the inference rules can be applied to the queries in parallel.

Instead of using the inference detection system to detect if a user has accessed particular data, we can also employ it as an anomaly detection system. For example, when a user has inferred certain amount of data in the database, it is reported to the security officer so that closer monitor to user activities will be carried out.

Acknowledgements The research reported in this paper is supported by the National Security Agency University Research Program under Contract DOD MDA904-96-1-0117, and by the CIA Office of Research and Development under Contract 96F 154000-000. The authors would like to acknowledge the helpful comments made by the reviewers of this paper.

References

[1] N. R. Adam and J. C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, December 1989.

[2] L. J. Binns. Inference through secondary path analysis. In B. M. Thuraisingham and C. E. Landwehr, editors, *Database Security VI: Status and Prospects*, pages 195–209. North-Holland, 1993.

[3] H. S. Delugach and T. H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Transactions on Data and Knowledge Engineering*, 8(1):56–66, 1996.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[5] J. Hale and S. Shenoi. Catalytic inference analysis: Detection inference threats due to knowledge discovery. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 188–199. IEEE Computer Society Press, 1997.

[6] T. H. Hinke. Inference aggregation detection in database management systems. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 96–106. IEEE Computer Society Press, 1988.

[7] T. H. Hinke and H. S. Delugach. Aerie: An inference modeling and detection approach for databases. In B. M. Thuraisingham and C. E. Landwehr, editors, *Database Security VI: Status and Prospects*, pages 179–193. North-Holland, 1993.

[8] T. H. Hinke, H. S. Delugach, and A. Chandrasekhar. Layered knowledge chunks for database inference. In T. F. Keefe and C. E. Landwehr, editors, *Database Security VII: Status and Prospects*, pages 275–295. North-Holland, 1994.

[9] T. H. Hinke, H. S. Delugach, and R. Wolf. A framework for inference-directed data mining. In P. Samarati and R. S. Sandhu, editors, *Database Security X: Status and Prospects*, pages 229–239. Chapman and Hall, 1997.

[10] T. H. Hinke, H. S. Delugach, and R. P. Wolf. Iliad: An integrated laboratory for inference analysis and detection. In S. A. D. David L. Spooner and J. E. Dobson, editors, *Database Security IX: Status and Prospects*, pages 333–348, 1995.

[11] T. F. Lunt. Aggregation and inference: Facts and fallacies. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 102–109. IEEE Computer Society Press, 1989.

[12] D. G. Marks. Inference in mls database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):46–55, February 1996.

[13] A. Motro, D. G. Marks, and S. Jajodia. Aggregation in relational databases: Controlled disclosure of sensitive imformation. In *Proceedings of the Third European Symposium on Research in Computer Security*, pages 431–445, November 1994.

[14] X. Qian, M. E. Stickel, P. D. Karp, T. F. Lunt, and T. D. Garvey. Detection and elimination of inference channels in multilevel relational database systems. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pages 196–205. IEEE Computer Society Press, 1993.

[15] M. E. Stickel. Elimination of inference channels by optimal upgrading. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 168–174. IEEE Computer Society Press, 1994.

[16] M. E. Stickel, T. D. Garvey, T. F. Lunt, and X. Qian. Inference channel detection and elimination in knowledge-based systems. Technical report, SRI International, Octobet 1994.

[17] T.-A. Su and G. Ozsoyoglu. Data dependencies and inference control in multilevel relational database systems. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 202–211. IEEE Computer Society Press, 1987.

[18] B. Thuraisingham. The use of conceptual structures for handling the inference problem. In C. E. Landwehr and S. Jajodia, editors, *Database Security V: Status and Prospects*, pages 333–362. Elsevier Science Pub. Co., 1992.