16 THE DESIGN AND IMPLEMENTATION OF A DATA LEVEL DATABASE INFERENCE DETECTION SYSTEM

Raymond W. Yip and Karl N. Levitt

Abstract: Inference is a way to subvert access control mechanisms of database systems. Most existing work on inference detection relies on analyzing functional dependencies in the database schema. This paper is an extension to our earlier effort in developing a data level inference detection system [13]. In this paper, we introduce the split query inference rule, make an extension to the overlapping inference rule, and provide an in depth discussion on the applications of the inference rules on union queries. Data level inference detection is inevitably expensive. We have developed a prototype of the inference detection system to evaluate its performance. The result shows that the system performs better with larger number of attributes and records in the database, and smaller number of projected attributes and return tuples of the queries. Therefore, the inference detection system could be practical when users retrieve a small amount of data compare to the size of the database.

16.1 INTRODUCTION

An inference occurs when a user infers data that the user is not allowed to access. In multilevel secure database systems, early work on inference detection employs a graph to represent the functional dependencies among attributes in the database schema. An inference occurs if there exists two paths between two attributes (or composite attributes), and the two paths are labeled at different

S. Jajodia (ed.), *Database Security XII*

classification levels [5, 1, 10]. The detected inference channel is eliminated by redesigning the database schema [8] or upgrading the paths that lead to the inference [11]. There is also work on incorporating external knowledge in detecting inference [12, 6, 3]. Detecting inference at the schema level is efficient as the detection is performed at the database design time. However, it has two drawbacks. First, the database schema does not capture all dependencies that occur in an instance of the database. Second, the existence of inference paths in the database schema does not necessary imply the users are making use of them to perform inference.

More recently, researchers look at the instance of the database to generate a richer set of functional dependencies for detecting inference. Hinke *et al.* use cardinality associations to discover potential inference channels [7]. Hale *et al.* incorporate imprecise and fuzzy database relations into their inference channel detection system [4]. Marks develops an inference detection system that prevents all possible inference by monitoring user queries with select clauses of the form "$A_i = a_i$", where a_i is a constant [9]. Chang *et al.* use Bayesian estimation and network techniques to estimate missing data in the database [2].

In this paper, we describe our effort in developing a data level inference detection system. We have identified six inference rules that users can use to infer data: split query, subsume, unique characteristic, overlapping, complementary, and functional dependence inference rules. Essentially, the six inference rules cover the set-subset, intersection, difference and union relationships among return tuples of queries. These rules are sound and they can be applied in any number of times, and in any order. The existence of these inference rules illustrates the inadequacy of the schema level inference detection approach.

However, data level inference detection is inevitably expensive, as it needs to keep track of all user queries and their return tuples. We have developed a prototype of the data level inference detection system to evaluate its performance. An earlier version of this paper is reported in [13]. In this paper, we introduce the split_query inference rule, make an extension to the overlapping inference rule, provide a detail description on the applications of the inference rules on union queries, and present a more complete experimental results. Because of lack of space, we omit the description of the unique characteristic and functional dependency inference rules. We also omit the use of examples to illustrate the inference rules. Interested readers can find them in [13].

This paper is organized as follows. In Section 2, we introduce the notations used in this paper. In Section 3, we present the inference rules. In Section 4, we discuss the applications of the inference rules on union queries. In Section 5, we outline the inference detection algorithm. In Section 6, we present our experimental results. In Section 7, we give a summary of the paper.

16.2 NOTATIONS

We consider a relational database that contains a single table. Multiple tables can be modeled as a universal relation as discussed in [9]. $t[A_i]$ denotes the

attribute value of the tuple t over the attribute A_i. A query is represented by a 2-tuple: *(projected-attributes; selection-criterion)*, where *projected-attributes* is the set of attributes projected by the query, and *selection-criterion* is the logical expression that selects the return tuples of the query. No aggregation function (for example, maximum and average) is allowed to apply on the projected-attributes. Given a query Q_i, $|Q_i|$ denotes the number of return tuples of Q_i, and $\{Q_i\}$ denotes the set of return tuples of Q_i. Unless otherwise stated, a set of return tuples is indeed a multiset of return tuples, that is, duplicated return tuples are retained. For each query $Q_i = \{AS_i; SC_i\}$, AS_i is expanded with A_i when '$A_i = a_i$' appears in SC_i as a conjunct. An *inferred query* is a query that a user can infer its return tuples without directly issuing it to the database. A *partial query* Q_i is a query that a user knows about $|Q_i|$ but not all the return tuples of Q_i. '$\cap$', '$\cup$', and '$\setminus$' stand for the set intersection, union, and difference operations respectively.

A tuple t projected over a set of attributes S *satisfies* a logical expression E if E is evaluated to *true* when each occurrence of A_i in E is replaced with $t[A_i]$, for all A_i in S. t *contradicts* E if E is evaluated to *false*. A return tuple t_i of a query Q_i is *indistinguishable* from another return tuple t_j of Q_j if 1) $t_i[A] = t_j[A]$ for each attribute $A \in (AS_i \cap AS_j)$, 2) t_i does not contradict SC_j, and 3) t_j does not contradict SC_i. A tuple t_i *relates* to another tuple t_j if the two tuples are projected from the same tuple in the database. If t_i relates to t_j, then t_i is indistinguishable from t_j; but the reverse does not necessary hold. Given two queries, Q_1 and Q_2, we say that Q_1 is *subsumed* by Q_2, denoted as $Q_1 \sqsubset Q_2$, if and only if 1) SC_1 logically implies SC_2 (denoted as $SC_1 \Rightarrow SC_2$), or 2) for each return tuple t_1 of Q_1, t_1 satisfies SC_2. '$\sqsubset$' is a reflexive, anti-symmetric, and transitive relation.

The goal of our inference detection system is to detect if a user can infer data using a series of queries. In particular, the system determines if a user can infer a return tuple of a query relates to a return tuple of another query. If so, the user can learn more about the return tuples.

16.3 INFERENCE RULES

In this section, we present four inference rules. Unless otherwise stated, all queries appear in the inference rules are not partial queries. We assume all the queries are issued by a single user, and there is no change to the database content. When two users are suspected of cooperating in performing inference, we run the inference detection system against their combined set of queries.

16.3.1 Split Queries

A query Q_i can be split into two smaller queries when a user can identify the return tuples of Q_i that relate to the return tuples of another query.

Inference Rule 1 (Split Queries) Given two queries Q_1 and Q_2. Express SC_2 in disjunctive normal form. If there exists a disjunct of SC_2 such that the set of attributes appear in the disjunct is a subset of AS_1, then generate two

inferred queries: 1) $Q_{11} = (AS_1; SC_1 \wedge SC_2)$; and 2) $Q_{12} = (AS_1; SC_1 \wedge \neg SC_2)$. Q_2 may be a partial query. The return tuples of Q_{11} are the return tuples of Q_1 that also satisfy SC_2. The return tuples of Q_{12} are the return tuples of Q_1 that does not satisfy SC_2.

When Q_1 projects all attributes that appear in a disjunct of SC_2, a user can identify the return tuples of Q_1 that satisfy SC_2. Hence, the user can divide the return tuples of Q_1 into two sets: those that satisfy both SC_1 and SC_2, and those that satisfy SC_1 but not SC_2.

16.3.2 Subsume Inference

In this section, we describe inference making use of the subsume relations among queries.

Inference Rule 2 (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 a return tuple t_1 of Q_1 is indistinguishable from exactly 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 $|S| = (|Q_2| - |Q_1|)$, generate two inferred queries from Q_2: 1) $Q_{21} = (AS_2;\ SC_2 \wedge \neg\ SC_1)$ with S as the set of return tuples; and 2) $Q_{22} = (AS_2;\ SC_2 \wedge SC_1)$ with $(\{Q_2\} \setminus S)$ as the set of return tuples. If $|S| < (|Q_2| - |Q_1|)$, generate an inferred partial query: $Q_{23} = (AS_2;\ SC_2 \wedge \neg\ SC_1)$ with S as the partial set of return tuples, and $|Q_{23}| = (|Q_2| - |Q_1|)$.

$Q_1 \sqsubset Q_2$ 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, a user can infer that each return tuple of Q_1 also takes the attribute value a over the attribute A. This is because for each return tuple t_1 of Q_1, no matter which return tuple t_2 of Q_2 that relates to t_1, $t_2[A] = a$. Hence, $t_1[A]$ must be equal to a.

SI2 says that if t_1 of Q_1 is indistinguishable from exactly one return tuple t_2 of Q_2, then t_1 relates to t_2. This is because $Q_1 \sqsubset Q_2$ implies that there is at least one return tuple of Q_2 that is indistinguishable from each return tuple of Q_1. Now, if t_1 of Q_1 is indistinguishable from one and only one return tuple t_2 of Q_2, then we can conclude that t_1 relates to t_2.

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 from Q_2: *(AS_2; $SC_1 \wedge SC_2$)* which includes return tuples of Q_2 that relate to the return tuples of Q_1, 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.

16.3.3 Overlapping Inference

In this section, we describe the overlapping inference rule.

Inference Rule 3 (Overlapping)

OI1 Given $Q_1 \sqsubset Q_2$, and $Q_1 \sqsubset Q_3$. Let S_2 be the set of return tuples of Q_2 that are indistinguishable from the return tuples of Q_3. If $|S_2| = |Q_1|$, and a return tuple t_2 of Q_2 is indistinguishable from exactly one return tuple t_3 of Q_3, then t_2 relates to t_3. Similarly, let S_3 be the set of return tuples of Q_3 that are indistinguishable from the return tuples of Q_2. If $|S_3| = |Q_1|$, and a return tuple t_3 of Q_3 is indistinguishable from exactly one return tuple t_2 of Q_2, then t_3 relates to t_2. Suppose $|Q_1| = |S_2| = |S_3|$. If a return tuple t_1 of Q_1 is indistinguishable from exactly one return tuple t_2 in S_2, then t_1 relates to t_2. Also, if t_1 is indistinguishable from exactly one return tuple t_3 in S_3, then t_1 relates to t_3. Q_1 may be a partial query.

OI2 Given a query Q_1, and a set of queries, $QS = \{Q_2, \ldots, Q_n\}$, where $n \geq 3$. Suppose $SC_1 \Leftrightarrow (SC_2 \vee \ldots \vee SC_n)$, and for each Q_i in QS, $Q_i \sqsubset Q_1$. If the number of distinguishable tuples in $QS = |Q_1|$, then any pair of indistinguishable tuples relate to each other.

OI3 When *OI1* is applied and all the related return tuples between Q_2 and Q_3 have been identified, generate the following two inferred queries from Q_2: 1) $Q_{21} = (AS_2; SC_2 \wedge \neg SC_3 \wedge \neg SC_1)$ with $\{Q_2\} \backslash S_2$ as the set of return tuples; and 2) $Q_{22} = (AS_2; SC_2 \wedge SC_3)$ with S_2 as the set of return tuples. Similarly generate two inferred queries from Q_3. When *OI2* is applied, generate possibly four inferred queries for each pair of queries that have overlapping return tuples.

Given that $Q_1 \sqsubset Q_2$ and $Q_1 \sqsubset Q_3$, the number of return tuples of Q_2 that relate to return tuples of Q_3 must be at least $|Q_1|$. *OI1* identifies the cases where a user can infer the related return tuples among the three queries. When Q_1 implies three or more queries, *OI1* is applied to two of them at a time.

We illustrate *OI2* using three queries, Q_1, Q_2, and Q_3, where $Q_1 \sqsubset Q_3$, $Q_2 \sqsubset Q_3$, and $SC_3 \Leftrightarrow SC_1 \vee SC_2$. Let N be the number of indistinguishable tuples in Q_1 and Q_2. As $SC_3 \Leftrightarrow SC_1 \vee SC_2$, each return tuple of Q_3 relates to a return tuple in Q_1 or Q_2. Hence, $N \geq |Q_3|$. Furthermore, as $Q_1 \sqsubset Q_3$ and $Q_2 \sqsubset Q_3$, each distinguishable tuple in Q_1 and Q_2 relates to a return tuple of Q_3. Hence, $N \leq |Q_3|$. Therefore, $N = |Q_3|$. When a user find out that the number of indistinguishable tuples in Q_1 and Q_2 equals $|Q_3|$, 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.

16.3.4 Complementary Inference

The complementary inference rule performs inference by eliminating tuples that are not related 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 Q_1 that relate to the return tuples of Q_3 are identified (for example using the overlapping inference rule), and the return tuples of Q_2 that relate to the return tuples of Q_4 are identified. 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 $Q'_2 = (AS_2; SC_2 \wedge \neg SC_4)$. $\{Q'_1\}$ is the set of return tuples of Q_1 that do not relate to any return tuple of Q_3, and $\{Q'_2\}$ is 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 $\{Q'_1\} \subset \{Q_1\}$, each return tuple of Q'_1 relates to a return tuple of Q_2. Condition (1) says that each return tuple of Q'_1 does not relate to any return tuple of Q_4. Hence, each return tuple of Q'_1 relates to a return tuple of Q'_2. Condition (2) or (3) implies $((Q_3 \sqsubset Q_4) \wedge (Q_4 \sqsubset Q_3))$. By removing from Q_1 and Q_2 the "same" set of return tuples, we have $Q'_1 \sqsubset Q'_2$.

It should be noted that in some cases, the inference as obtained from the complementary inference rule can also be obtained from the overlapping inference rule. For example, consider four queries Q_1, Q_2, Q_3, and Q_4, where $Q_1 \sqsubset Q_2$, and $Q_3 \sqsubset Q_4$. Suppose the overlapping inference rule can be applied to identify the related tuples between Q_1 and Q_3, and between Q_2 and Q_4. These result in the generation of two inferred queries: 1) $Q'_1 = (AS_1; SC_1 \wedge \neg SC_3)$; and 2) $Q'_2 = (AS_2; SC_2 \wedge \neg SC_4)$. If $(SC_1 \wedge \neg SC_3) \Rightarrow (SC_2 \wedge \neg SC_4)$, then we have $Q'_1 \sqsubset Q'_2$ which is the same result as obtained by applying the complementary inference rule to the four queries. However, $SC_1 \Rightarrow SC_2$ and $SC_3 \Rightarrow SC_4$ does not necessary implies $(SC_1 \wedge \neg SC_3) \Rightarrow (SC_2 \wedge \neg SC_4)$. When this implication does not hold, the complementary inference rule is needed to perform the inference.

16.4 INFERENCE WITH UNION QUERIES

The inference rules can be applied to unions of queries. We call a union of queries a *'union query'*. In contrast, a user query or an inferred query is called a *'simple query'*. If Q_u is a union query consists Q_i, ..., and Q_j, then $AS_u = (AS_i \cap \ldots \cap AS_j)$, and $SC_u = (SC_i \vee \ldots \vee SC_j)$. Note that AS_u might be equal to $\emptyset$. The applications of the split query, unique characteristic and functional dependency inference rules on union queries are similar to their applications on simple queries. Hereafter, we only discuss the applications of the subsume, overlapping, and complementary inference rules on union queries.

16.4.1 Subsume Inference Rule on Union Queries

Consider the applications of the subsume inference rule on union queries when the union queries are subsumed by other queries. Let $Q_u = \{Q_i, \ldots, Q_j\}$ be a union query, and $Q_u \sqsubset Q_1$. We show that inference obtained by applying the subsume inference rule on $(Q_i \cup \ldots \cup Q_j) \sqsubset Q_1$ can also be obtained by applying the subsume inference rule on $Q_i \sqsubset Q_1$, ..., and $Q_j \sqsubset Q_1$.

Consider the applications of *SI1*. If there is an attribute A in $(AS_1 \backslash AS_u)$, such that all return tuples of Q_1 take the same attribute value a over A, then for each return tuple t_u of Q_u, $t_u[A] = a$. This implies that for each return tuple t of a simple query of Q_u, $t[A] = a$. This is the same as if the *SI1* is applied to Q_i and Q_1, where $Q_i \sqsubset Q_1$, for each simple query Q_i of Q_u.

Consider the applications of *SI2*. If there exists a tuple t_u in Q_u that is indistinguishable from exactly one return tuple t_1 of Q_1, there exists at least one simple query Q_i of Q_u such that t_u relates to a return tuple t_i of Q_i. Now, t_i is indistinguishable from t_1 of Q_1. Hence, when *SI2* is applicable to infer that t_u of Q_u relates to t_1 of Q_1, it is also applicable to infer that t_i of Q_i relates to t_1 of Q_1.

Consider the applications of *SI3*. When all the related tuples between Q_u and Q_1 are identified, two inferred queries are generated from Q_1: 1) $Q_{u1} = (AS_1; SC_1 \wedge \neg SC_u)$; and 2) $Q_{u2} = (AS_1; SC_1 \wedge SC_u)$. We show that these two queries can also be generated from the simple queries of Q_u and Q_1. Note that when all the related tuples between Q_u and Q_1 have been identified, all related tuples among the simple queries of Q_u are also identified. Without loss of generality, suppose $Q_u = \{Q_2, Q_3\}$. The application of *SI3* on Q_1 and Q_2 generates two inferred queries: 1) $Q_{21} = (AS_1; SC_1 \wedge \neg SC_2)$; and 2) $Q_{22} = (AS_1; SC_1 \wedge SC_2)$. Similarly, the application of *SI3* on Q_1 and Q_3 generates two inferred queries: 1) $Q_{31} = (AS_1; SC_1 \wedge \neg SC_3)$; and 2) $Q_{32} = (AS_1; SC_1 \wedge SC_3)$. Now, Q_{21} and Q_{31} are both generated from Q_1, and we can generate the following inferred query for their related tuples: $(AS_1; SC_1 \wedge \neg SC_2 \wedge \neg SC_3)$ which equals Q_{u1}. Q_{22} and Q_{32} are both generated from Q_1, and we can identify the related tuple between them. The union of these two queries is $(AS_1; SC_1 \wedge (SC_2 \vee SC_3))$ which equals Q_{u2}. Therefore, we do not need to consider the applications of the subsume inference rule when the union query is subsumed by other queries.

Consider the case where union queries subsume other queries, say $Q_1 \sqsubset Q_u$. *SI1* is applied as follows. If for each return tuple t of any simple query of Q_u, $t[A] = a$, then $t_1[A] = a$ for each return tuple t_1 of Q_1. *SI2* is applied as follows. If there is a return tuple t_1 of Q_1 that is indistinguishable from a set of return tuples S from the simple queries of Q_u, where all tuples in S relate to one another, then t_1 relates to each tuple in S. *SI3* is applied similarly. Note that the subsume inference rule can still be applied when the simple queries of Q_u have no common projected attribute.

16.4.2 *Overlapping and Complementary Inference Rule on Union Queries*

Consider the applications of *OI1*. Given three queries, Q_1, Q_2, and Q_u, where Q_u is a union query. Suppose $Q_u \sqsubset Q_1$ and $Q_u \sqsubset Q_2$. If *OI1* is to be applied to identify the related return tuples among Q_2 and Q_3, $|Q_u|$ must be known. That is, the number of related tuples, if any, between the simple queries are identified. Now, suppose $Q_1 \sqsubset Q_u$ and $Q_1 \sqsubset Q_2$. If *OI1* is to be applied to identify the related return tuples between Q_u and Q_2, then the user must has already identified those related tuples among the simple queries in Q_u. Also, the user has to identify the return tuples of Q_u that are indistinguishable from the return tuples of Q_2, and the number of these return tuples equals $|Q_1|$.

Consider the applications of *OI2*. Suppose there is a set of queries $QS = \{Q_2, \ldots, Q_n, Q_u\}$ such that for each query $Q_i \in QS$, $Q_i \sqsubset Q_1$. *OI2* is applicable when the related tuples among the queries in QS are identified. That is, the related return tuples, if any, between Q_u and other queries in QS have to be identified. *OI3* is applied similar to the case with simple queries.

Note that the overlapping inference rule can still be applied when $AS_u = \emptyset$. For example, let $Q_u = \{Q_{u1}, Q_{u2}\}$. If $SC_{u1} \wedge SC_{u2} = false$, the user can conclude that there is no related return tuple between Q_{u1} and Q_{u2}, and $|Q_u| = |Q_{u1}| + |Q_{u2}|$.

Consider the applications of the complementary inference rule on the union queries. Suppose there are four queries Q_1, Q_2, Q_3, and Q_u, where Q_u is a union query, $Q_1 \sqsubset Q_2$, and $Q_3 \sqsubset Q_u$. To apply the complementary inference rule on these four queries, the related return tuples among the simple queries in Q_u that also relate to return tuples of Q_2 must have been identified. Similarly for the case when Q_1, Q_2, or Q_3 is a union query.

16.5 INFERENCE DETECTION ALGORITHMS

In this section, we outline the inference detection algorithms. Figure 16.1 shows the main function *INFERENCE(U, Q_i)*, which is called each time a user U issues a query Q_i to the database. The function maintains two sets: *GEN* and *EXP*. *GEN* is initialized with the user issued query Q_i, and is subsequently being added with inferred queries generated by the inference rules. Each query in *GEN* is compared with previously issued or inferred queries for user U (denoted as *PREV_QUERY(U)*) to determine if the inference rules are applicable to them. *EXP* is the set of tuples that are expanded during the applications of the inference rules. The results of the applications of inference rules are generations of inferred queries and expansions of some return tuples of queries. Given a tuple t_1 projected over a set of attributes AS_1, and another tuple t_2 projected over a set of attributes AS_2. If t_1 and t_2 are found to be related to each other, t_1 is expanded as follows: for each attribute $A \in AS_2 \backslash AS_1$, $t_1[A] = t_2[A]$. t_2 is expanded similarly.

After a tuple is expanded, the query that returns the expanded tuple might be eligible in further applications of inference rules. Hence, the function checks if the inference rules are applicable to the query. *INFERENCE* is a terminating

function, as the number of inferences is bound by the size of the database. In each call to the *INFERENCE* function, all queries in *GEN* are processed before the expanded tuples in *EXP*. This avoids repeatedly processing the same tuple which is expanded more than once after queries in *GEN* are processed.

```
INFERENCE (U, Q_i):
1.    initialize GEN with Q_i;
2.    EXP ← ∅;
3.    GEN_Q ← ∅;
4.    EXP_Q ← ∅;
5.    while (GEN ≠ ∅ or EXP ≠ ∅) do
6.       if GEN ≠ ∅ then
7.          Q_j ← a query in GEN;
8.          remove Q_j from GEN
9.          GEN_Q ← GEN_Q ∪ {Q_j};
10.      else if EXP ≠ ∅ then
11.         Q_j ← a query that returns a tuple in EXP;
12.         EXP_Q ← EXP_Q ∪ {Q_j};
13.      ts ← return tuples of Q_j in EXP;
14.      remove return tuples of Q_j from EXP;
15.      for each Q_k ∈ PREV_QUERY(U) do
16.         EXP ← UNIQUE(Q_j, Q_k, ts, EXP);
17.         GEN ← SPLIT_QUERY(Q_j, Q_k, GEN);
18.         if Q_j ⊏ Q_k then
19.            (GEN, EXP) ← SUBSUME(Q_j, Q_k, GEN, EXP);
20.            (GEN, EXP) ← OVERLAP(U, Q_j, Q_k, GEN, EXP);
21.            GEN ← COMPLEMENTARY(Q_j, Q_k, GEN);
22.         else if Q_k ⊏ Q_j then
23.            (GEN, EXP) ← SUBSUME(Q_k, Q_j, GEN, EXP);
24.            (GEN, EXP) ← OVERLAP(U, Q_k, Q_j, GEN, EXP);
25.            GEN ← COMPLEMENTARY(Q_k, Q_j, GEN);
26.   FIND_UNION(U, GEN_Q, EXP_Q);
```

Figure 16.1 The inference function.

The function *UNIQUE* has three input parameters: Q_j, Q_k, and ts. The function checks if unique characteristic can be determined between the two queries Q_j and Q_k. For each expanded return tuple in ts, the function checks if the expanded return tuple and another return tuple have common unique characteristics. If so, the two return tuples are expanded with each other. The functions *SPLIT_QUERY*, *SUBSUME*, *OVERLAP*, and *COMPLEMENTARY* operate as described in the corresponding inference rules, and we omit the presentations of their algorithms. The *FIND_UNION* function checks if there are unions of query that satisfy the subsume relations with other queries. If so, the inference rules are applied to them.

16.6 EXPERIMENTAL RESULTS

We have developed a prototype of the inference detection system in about 4,000 lines of Perl code. We have implemented the split query, subsume, unique characteristic, overlapping (except *OI2*), and complementary inference rules. The system also handles applications of the inference rules on union queries. 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. Attribute values in the table are 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 comparison operators ($>$, $\geq$, $\leq$, $<$, and $=$), and a_i is an attribute value. Each query has N_{ret_tuple} number of return tuples. 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 appear in C_j is a subset of those appear in C_i. We collect the following two data to measure the system performance: 1) average number of seconds used to process one query. 2) number of times the inference rules are applied.

We ran six experiments to determine how the characteristics of the database and the queries affect the system performance. For the database, we consider the following characteristics: 1) the number of tuples in the database; 2) the number of attributes in the database; and 3) the amount of duplication of the data values. For the queries, we consider the following characteristics: 1) the number of attributes projected by the queries; 2) the number of conjuncts in the selection criteria; 3) the number of queries being issued; and 4) the number of tuples returned by the queries. The experimental results of running the inference detection system on a Sun SPARC 20 workstation are shown in Figure 16.2–Figure 16.7.

Experiment 1 investigates the effect of the number of attributes and the amount of data duplication in the database on the system performance. In this experiment, we choose the following parameter values: $N_{rec\_num} = 1000$, $N_{ret\_tuple} = 50$, $N_{proj} = 4$, $N_{cond} = 3$, and $N_{query\_num} = 500$. N_{attr} is varied with the following values: 40, 60, 80, 100, 120, and 140. N_{data_dist} is varied with the following values 25%, 50%, 75%, and 100%. Figure 16.2 shows the results in a graph plotted with the average query processing time (in seconds) against the number of attributes in the database. Consider each individual line in Figure 16.2. It shows that the system runs faster as N_{attr} increases from 40 to 140. With a fixed type of queries, the larger the number of attributes in the table, the lesser the amount of overlapping among the return tuples of queries. This results in lesser subsume relations hold among queries, and hence the smaller the number of inferences.

Consider the four lines in Figure 16.2. They correspond to the cases where N_{data_dist} = 25%, 50%, 75%, and 100%. The lower the value of N_{data_dist}, the more duplication of the data in the database. Intuitively, the higher the duplication of the data, the lesser the number of distinguishable return tuples, and hence the smaller number of inferences. This is ture in some cases. However, in general the results do not show a significant effect of data duplication on the system performance.

Experiment 2 investigates the effect of the number of return tuples of queries on the system performance. Figure 16.3 shows the results for N_{rec_num} = 1000, N_{data_dist} = 50%, N_{proj} = 4, N_{cond} = 3, and N_{query_num} = 500. N_{ret_tuple} takes the values of 50, 100, 150, 200, and 250, and N_{attr} takes the values of 80 and 120. The figure shows that the system runs slower as N_{ret_tuple} increases. The larger the number of return tuples, the longer it takes for the system to process them. Also, the more the number of tuples returned by the queries, the more the number of occurrences of inferences, and also the more the number of inferred queries being generated.

Experiment 3 investigates the effect of the number of projected attributes in queries on the system performance. Figure 16.4 shows the results for N_{rec_num} = 1000, N_{query_num} = 500, N_{data_dist} = 50%, N_{attr} = 80, and N_{ret_tuple} = 50. N_{proj} takes the values of 4, 5, 6, 7, and 8. N_{cond} takes the values of 4, 5, 6, and 7. It shows that the system runs slower as N_{proj} increases. This is because the more the number of attributes projected by the queries, the more overlapping among the return tuples of queries, and hence the more number of inferences.

Experiment 4 investigates the effect of the number of conjunts in the selection criteria on the system performance. Figure 16.5 shows the results for N_{rec_num} = 1000, N_{query_num} = 500, N_{data_dist} = 50%, N_{attr} = 80, and N_{ret_tuple} = 50. N_{cond} takes the values of 3, 4, 5, 6, and 7. N_{proj} takes the values of 4, 5, 6, and 7. It shows that the system runs faster as N_{cond} increases. This is because the larger the number of conjuncts in the selection criteria of the queries, the lesser the chance that the subsume relations hold among the queries, and hence the smaller number of occurrences of inferences. However, the effect is not significant when $N_{cond} > 3$.

Experiment 5 investigates the effect of the number of tuples in the database on the system performance. Figure 16.6 shows the result for N_{data_dist} = 50%, N_{attr} = 80, N_{ret_tuple} = 50, N_{query_num} = 500, N_{proj} = 4, and N_{cond} = 3. N_{rec_num} is varied with the following values: 1000, 2500, 5000, 7500, and 10000. It shows that the system runs faster as the number of tuples of the database increases. As the size of the database increases, the possible amount of overlapping among the queries decreases, and hence the lesser number of inferences. For N_{ret_tuple} = 10000, the set of queries happen to generate more inferences than the case for N_{ret_tuple} = 5000 or 7500, and hence it has a longer running time.

Experiment 6 investigates the effect of the number of queries on the system performance. Figure 16.7 shows the results for N_{rec_num} = 1000, N_{data_dist} = 50%, N_{attr} = 80, N_{ret_tuple} = 30, N_{proj} = 4, and N_{cond} = 3. N_{rec_number} takes

the values of 200, 400, 600, 800, 1000, and 1200. It shows that the system runs slower as the number of queries to be processed increases. This is because the more the number of queries, the more the number of inferences. Also, as each user query needs to be compared with previously issued queries for the subsume relations, the more the number of queries, the longer it takes to determine all possible subsume relations.

16.7 SUMMARY

In this paper, we describe our effort in developing a data level inference detection system. We have identified six inference rules: split query, subsume, unique characteristic, overlapping, complementary, and functional dependency inference rules. We have also discussed the applications of the inference rules on union queries. The existence of these inference rules shows that simply using functional dependencies to detect inference is inadequate. We have developed a prototype of the inference detection system using Perl on a Sun SPARC 20 workstation.

Although the data level inference detection approach is inevitably expensive, there are cases where the uses of such approach is practical. As shown in our experimental results, the system generally performs better with a larger size of the database, and queries that return smaller number of tuples and project smaller number of attributes. The system running time becomes high when queries retrieve a large amount of data from the database, and there are large amount of overlapping among query results. However, when a user issues such type of queries, it is suspicious that the user is attempting to infer associations among the data.

Acknowledgments

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.

References

[1] Leonard J. Binns. Inference through secondary path analysis. In *Proc. 6th IFIP WG11.3 Workshop on Database Security*, pages 195–209, August 1992.

[2] LiWu Chang and Ira S. Moskowitz. Bayesian methods applied to the database inference problem. In *Proc. 12th IFIP WG11.3 Workshop on Database Security*, July 1998.

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

[4] John Hale and Sujeet Shenoi. Catalytic inference analysis: Detection inference threats due to knowledge discovery. In *Proceedings of the 1997*

IEEE Symposium on Research in Security and Privacy, pages 188–199, May 1997.

[5] Thomas H. Hinke. Inference aggregation detection in database management systems. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 96–106, April 1988.

[6] Thomas H. Hinke, Harry S. Delugach, and Asha Chandrasekhar. Layered knowledge chunks for database inference. In *Proc. 7th IFIP WG11.3 Workshop on Database Security*, pages 275–295, September 1993.

[7] Thomas H. Hinke, Harry S. Delugach, and Randall Wolf. A framework for inference-directed data mining. In *Proc. 10th IFIP WG11.3 Workshop on Database Security*, pages 229–239, July 1996.

[8] Teresa F. Lunt. Aggregation and inference: Facts and fallacies. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 102–109, May 1989.

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

[10] Xiaolei Qian, Mark E. Stickel, Peter D. Karp, Teresa F. Lunt, and Thmoas D. Garvey. Detection and elimination of inference channels in multilevel relational database systems. In *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy*, pages 196–205, May 1993.

[11] Mark 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, May 1994.

[12] Bhavani Thuraisingham. The use of conceptual structures for handling the inference problem. In *Proc. 5th IFIP WG11.3 Workshop on Database Security*, pages 333–362, November 1991.

[13] Raymond Yip and Karl Levitt. Data level inference detection in database systems. In *Proc. 11th IEEE Computer Security Foundations Workshop*, pages 179–189, June 1998.

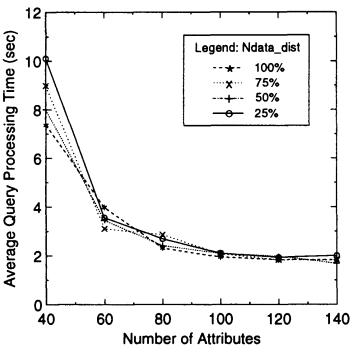

Figure 16.2 Effect of the Number of Attributes in the Database.

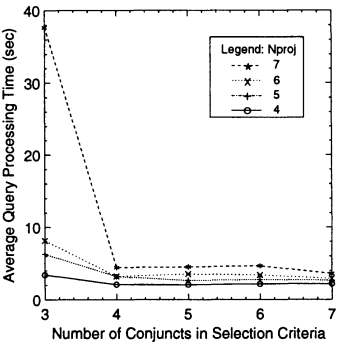

Figure 16.5 Effect of the Size of the Selection Criteria of Queries.

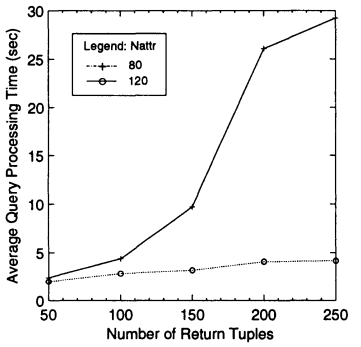

Figure 16.3 Effect of the Number of Return Tuples of Queries.

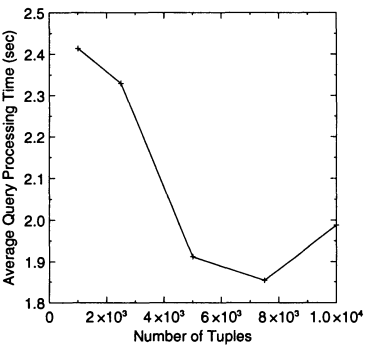

Figure 16.6 Effect of the Number of Tuples in the Database.

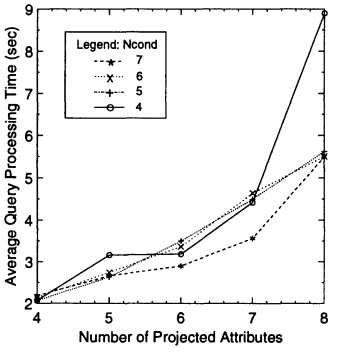

Figure 16.4 Effect of the Number of Projected Attributes of Queries.

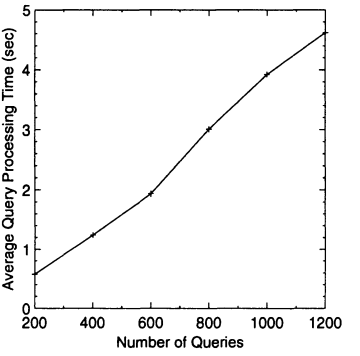

Figure 16.7 Effect of the Number of Queries Issued.