Discovery and Maintenance of Functional Dependencies by Independencies

Siegfried Bell

Informatik VIII
University Dortmund
44221 Dortmund Germany
bell@ls8.informatik.uni-dortmund.de

Abstract

For semantic query optimization one needs detailed knowledge about the contents of the database. Traditional techniques use static knowledge about all possible states of the database which is already given. New techniques use knowledge only about the current state of the database which can be found by methods of knowledge discovery in databases. Databases are often very large and permanently in use. Therefore, methods of knowledge discovery are only allowed to take a small amount of the capacity of the database system. So database access has to be reduced to a minimum during the process of discovery and obviously if the database changes during the process of maintenance of the discovered knowledge.

In this paper, the main effort has been put into minimizing the number of database accesses, w.r.t. discovery and maintenance. This is exemplified by the discovery of functional dependencies. We improve the inference of functional dependencies by using independencies and cardinality dependencies. First we give an axiomatization. Then we describe the three parts of our system: the initialization by cardinality dependencies, the exclusion of hypothetical dependencies by entailment, and the verification of hypothetical dependencies by SQL queries. Moreover, we investigate the complexity of each subsystem. In addition, we describe the maintenance of discovered dependencies and finally draw some conclusions.

Introduction and Related Work

Query optimization can be regarded as the process of transforming a query Q into another query Q' that can be evaluated more efficiently, as mentioned by Chakravarthy et al. (Chakravarthy, Grant, & Minker 1990). Semantic query optimization (SQO) is mainly based on the use of semantic knowledge during the optimization process. Thus, the user is motivated to concentrate on the application rather than forming queries with explicit semantic knowledge of the application.

So far, the main problem is to provide the optimizers with semantic knowledge about the database during SQO. Obviously, the only kind of semantic knowledge which is always available in relational databases management systems (DBMS) are integrity constraints like primary or foreign keys. Thus, Chakravarthy et al. (Chakravarthy, Grant, & Minker 1990) have defined SQO in respect of integrity constraints as to transform a query into one which is semantically equivalent to the original query, but which can be executed more efficiently. King (King 1981) argued that the costs of evaluating the transformed query plus the transformation costs should be lower than the costs of evaluating the original query. Semantic equivalence means that the transformed query has the same answer as the original query on all database states satisfying the integrity constraints. Jarke et al. (Jarke, Clifford, & Vassiliou 1984) have shown several ways to use functional dependencies for SQO. But the constraints provided by a DBMS are few and often too general in the sense that they are valid in all possible database states.

Another way is to provide SQO with semantic knowledge by hand which also seems no adequate technique. For example, King (King 1981) uses constraints on attribute values to optimize queries by his system QUIST. Zhang and Ozsoyoglu (Zhang & Ozsoyoglu 1994) have presented also techniques for semantic query optimization which are based on implication constraints. Implication constraints are a generalization of functional dependencies.

The arise of knowledge discovery in databases (KDD) offers a new approach to solve both problems: provides SQO automatically with constraints and extends them to constraints which precisely reflects the present content of the database. Siegel has reported this by the first time (Siegel 1988) and (Siegel, Sciore, & Salveter 1991). Such constraints have been termed, for example, *Database Abstractions* in (Hsu & Knoblock 1993), *Metadata* in (Siegel & Madnick 1991), and *Meta Knowledge* in (Schlimmer 1991). Also, Hsu and Knoblock (Hsu & Knoblock 1993) have shown the benefits of optimization techniques based on automatically discovered constraints. But we have to keep in mind that these constraints are only valid in the present state of the database and therefore describe the content of the database precisely. The constraints may become invalid, if the database changes. Therefore, we have to maintain the discovered knowledge, if we use it more than once.

Problems arise if knowledge discovery is applied to real world databases which are continuously in use and large. Therefore, knowledge discovery in databases is only allowed to take a small portion of the system resources.

Comparable to our approach in order to discover functional dependencies, there are similar ones by Mannila and Räihä (Mannila & Räihä 1991), Dehaspe et al. (Dehaspe, Laer, & Raedt 1994), Savnik and Flach (Savnik & Flach 1993), and Schlimmer (Schlimmer 1993). Mannila and Räihä have investigated the problem of inferring FDs from example relations in order to determine database schemes. But they do not use a complete inference relation regarding independencies. Savnik and Flach have investigated a special data structure for the FDs. Briefly, they start with a bottom-up analysis of the tuples and construct a *negative cover*, which is a set of FIs. In the next step they use a top-down search approach. They check the validity of a dependency by searching for FIs in the negative cover. Also, the negative cover is not complete regarding a classical consequence relation. Schlimmer also uses a top-down approach, but in conjunction with a hash-function in order to avoid redundant computations. However, he does not use a complete inference relation even regarding functional dependencies. Also do Dehaspe et al. because their inferences are based on Θ subsumption. In addition, the verification is based on theorem proving which is not suitable for real world databases.

In general, these authors do not use a relational database like OracleV7 or any other commercial DBMS. In such case, we argue that the proposed algorithm and approaches have to redesign according the set oriented interface of a relational database system. For example, the concept of the negative cover has only advantages if the tuples can be accessed directly, i.e. the tuples are stored in the main memory as Prolog-facts. Savnik and Flach have introduced it because the complexity for testing contradiction of the FDs is reduced.

In contrast to these appproaches our purpose is different because we maintain the discovered FDs in order to use them all the time by semantic query optimization.

In addition, we argue that by using a relational database system, the higher complexity of the complete inference relation is justified by the size of a real world database.

We have already investigated the discovery of value restrictions, unary inclusion dependencies, and functional dependencies in relational databases and has implemented this approach by using the database management system ORACLE (Bell & Brockhausen 1995). Also, we have given an approach to test these dependencies by SQL statements and have given an empirical evaluation.

$\stackrel{n}{=}$	1	NULL	2
1	true	false	false
NULL	false	true	false
2	false	false	true

Figure 1: Equality operator $\stackrel{n}{=}$

Thus, this paper focuses on minimizing the number of accesses to the database w.r.t. the discovery of functional dependencies and their maintenance. The inference of functional dependencies is improved by using functional independencies and by cardinality dependencies. Also, we regard `Null` values because they frequently arise in databases. We give an axiomatization of these independencies and discuss the complexity of the implemented inference relation which is given by three algorithms. Then we describe dependency maintenance caused by insertion or deletion of tuples.

Functional Independencies

In this section we discuss functional independencies and their axiomatization. We assume familiarity with the definitions of relational database theory (see for example (Kanellakis 1990)) and the basic properties of the classical consequence relation. The capital letters $A, B, C, \ldots$ denote attributes, and X, Y, Z denote attribute sets. We do not distinguish between an attribute A and an attribute set $\{A\}$. We omit proofs when they are trivial. Remember that every attribute is associated with a set of values, called its *domain*. Functional dependencies are defined as usual. Additionally, we consider `Null` values because the ISO standard permits `Null` values in any attribute of a *candidate key*. Therefore, we adopt a special equality operator for the definition of the FDs, $\stackrel{n}{=}$, which is illustrated in figure 1.

The so called Armstrong's Axioms provide a correct and complete axiomatization for them and establish an inference relation $\vdash_A$.

We now can define the consequences of a set of dependencies. Let Σ be a set of functional dependencies, then $X \rightarrow Y$ is a consequence of Σ, or $X \rightarrow Y \in Cn(\Sigma)$: whenever a relation satisfies Σ it satisfies $X \rightarrow Y$. The closure of attributes X w.r.t. a set of FDs Σ is defined as: $closure(X, \Sigma) = \{Y \mid X \rightarrow Y \in Cn(\Sigma)\}$ and is denoted by $\overline{X}$.

Functional *independencies* have been introduced by Janas (Janas 1988) to mirror functional dependencies. But they are meant for a totally different purpose: FIs are not semantical constraints on the data, but a support for the database designer in the task of identifying functional dependencies. In addition, they also help to improve the inference of functional dependencies. We simplify the definition of functional independencies here by ignoring `Null` values.

In (Paredaens *et al.* 1989) *afunctional dependencies*[1] are introduced, but they are a sort of semantic constraints and much stronger than our functional independencies. Therefore, we cannot use them to improve the inference of FDs.

Definition 1 (Functional Independency (FI)) *$X \not\to Y$ denotes a functional independency. A relation r satisfies $X \not\to Y$ ($r \models X \not\to Y$) if there exist tuples t_1, t_2 of r with $t_1[X] \overset{n}{=} t_2[X]$ and $t_1[Y] \overset{n}{\neq} t_2[Y]$.*

The consequences of FDs and FIs are defined as follows: Let Σ be a set of FDs and Σ' a set of FIs. $Cn(\Sigma \cup \Sigma') := \{\sigma |$ for each relation r if $r \models \Sigma \cup \Sigma'$, then $r \models \sigma\}$ where σ is a FD or FI. $\Sigma \cup \Sigma'$ is called inconsistent, if there exists a $X \not\to Y$ so that $X \to Y \in Cn(\Sigma \cup \Sigma')$ and $X \not\to Y \in Cn(\Sigma \cup \Sigma')$ or *vice versa.*

An axiomatization of FIs has already been given by Janas (Janas 1988) which establishes an inference relation $\vdash_{Janas}$.

Definition 2 (FIs) *The axiomatization by Janas is given by:*

1. $\frac{X \not\to Y}{X \not\to YZ}$
2. $\frac{XZ \not\to YZ}{XZ \not\to Y}$
3. $\frac{X \to Y, X \not\to Z}{Y \not\to Z}$

We show with a counterexample that this inference relation is not complete, i.e. there exists some $X \not\to Y$ with $X \not\to Y \in Cn(\Sigma \cup \Sigma')$ and $\Sigma \cup \Sigma' \not\vdash_{Janas} X \not\to Y$.

Lemma 1 *The following inference rule is correct:*

$$\frac{X \to Y, Z \not\to Y}{Z \not\to X}$$

Lemma 2 $\{X \to Y, Z \not\to Y\} \not\vdash_{Janas} Z \not\to X$.

Proof: We assume that X, Y and Z are disjoint. Then obviously the first and the second rule cannot be applied to infer $Z \not\to X$. Thus, the third rule can be applied only. But Z is not in the closure of Y and Σ, i.e. $\{X \to Y\} \not\vdash_A Y \to Z$. Thus, we cannot infer $Z \not\to X$. □

Theorem 1 *The axiomatization by Janas (Janas 1988) is not complete.*

Instead, we propose the following axiomatization:

Definition 3 (Inference of FIs) *An inference relation $\vdash_{fi}$ is given by an axiomatization of the FDs and the following inference rules:*

FI1: $\frac{XV \not\to YW, W \subseteq V}{X \not\to Y}$

FI2: $\frac{X \to Y, X \not\to Z}{Y \not\to Z}$

FI3: $\frac{Y \to Z, X \not\to Z}{X \not\to Y}$

For example, the functional independency $X \not\to YZ$ which is a consequence of Janas' first inference rule can be inferred by $\vdash_{fi}$ as follows: we infer $YZ \to Y$ by Armstrong's Axioms and use FI3 to infer a FI $X \not\to YZ$ from $X \not\to Y$ and $YZ \to Y$.

Theorem 2 (Soundness and Completeness) *The inference rules FI1 to FI3 are correct and complete for a consistent set of functional dependencies, respectively independencies.*

Proof: For details refer to (Bell 1995). A rough sketch is as follows: First, it has to be shown that the FIs do not affect the inference of FDs. Second, that we can infer from a set of FIs only trivial inferences, e.g. if $AB \not\to C$ then $A \not\to C$ or if $AB \not\to C$, then $AB \not\to CD$. The last part can be seen by a negative proof of Armstrong's Axioms, or by a construction of the certain possible cases. □

Inference of Functional Independencies

Our system consists of three elements: initialization, entailment, and verification. It is roughly sketched in table 1. First, we initialize our data structure `List` for the FDs and FIs. Then, we generate hypothetical dependencies, check if these are already entailed by the known dependencies or independencies, and verify the remaining ones by querying the database. We use a kind of breath first search because we generate only hypotheses which are not related by each other. Terminating is ensured, because if no already entailed hypotheses can be generated, then the algorithm stops. The algorithms are written in PROLOG and connected to an ORACLE7 server by a SQL interface via TCP/IP.

Verification

Functional dependencies can be verified by sorting the tuples of the relation which takes $O(n \log n)$ time w.r.t. the number of tuples, cf. (Mannila & Räihä 1991). In our implementation we use the **nvl** statement in SQL to handle the NULL values.

Entailment

Entailment of FDs is often discussed by studying if $\Sigma \vdash_A X \to Y$ holds where Σ is a set of FDs and $n = |\Sigma|$. This can be decided in linear time with appropriate data structures (Kanellakis 1990). By Lemma 2, we can easily construct an algorithm for testing FIs:

```
function $\Sigma \cup \Sigma' \vdash_{fi} X \not\to Y$;
begin
    for each $V \not\to W \in \Sigma'$ do
            if $\Sigma \vdash_A V \to X$ and $\Sigma \vdash_A Y \to W$
               then return Yes;
               else return No;
end;
```

[1]The definition of the AD $X \not\twoheadrightarrow Y$ requires that for each tuple t there exists a tuple t' so that $t[X] \overset{n}{=} t'[X]$ and $t[Y] \overset{n}{\neq} t'[Y]$.

```
1. Initialize List.
2. Repeat
   (a) Take an element $t$ from List and generate
       all tuples $T$ with a fixed length that are
       not entailed by List.
   (b) Query DB server for $T$.
   (c) Add $T$ to *List* and find a minimal cover
       for it.
3. until no already entailed hypothesis can be
   generated
```

Table 1: Description of our System

It is obvious that testing FIs takes $O(n^2)$ time where $n = max(\Sigma, \Sigma')$. Correctness and completeness follow immediately from the previous section.

The sets of FDs and FIs are usually very large. We can reduce these sets taking into account the following observation: The set of functional dependencies is partitioned into equivalence classes by the satisfiability definition. Each class of functional dependencies specifies the same set of admissible relations. As these equivalence classes will typically contain a large number of elements, it is reasonable to define a suitable representation with a minimal number of elements. This representation is usually called a minimal cover, see (Maier 1980). We can simply extend the definitions in (Maier 1980) by using our inference relation $\vdash_{fi}$:

Definition 4 (Minimal Cover) *Let Σ be a minimal set of FDs.*

Σ' is a set of FIs and is called minimal if for all $V \not\rightarrow W \in Cn(\Sigma \cup \Sigma')$ there exists no $X \not\rightarrow Y \in \Sigma'$ with $\Sigma \cup \Sigma' \backslash \{X \not\rightarrow Y\} \vdash_{fi} V \not\rightarrow W$.

Therefore, minimizing can be done by repeated application of $\vdash_{fi}$ and takes $O(n^3)$ time for some set of FDs and FIs.

Initialization

The data structures of the FDs and FIs are initialized with information about primary keys and sufficient conditions for FIs. These conditions are given by the cardinality of the attributes which were introduced by Kanellakis et al. as Unary Cardinality Dependencies (UCDs) in the unary case (Kanellakis, Cosmadakis, & Vardi 1983). We have extended these UCDs to the general case of CDs and have given an axiomatization in (Bell 1995). Thus, we propose the following two rules as sufficient conditions for FIs:

Definition 5 (Cardinality of Attributes) *Let X and Y be sets of attributes:*

CD-FI1: $\frac{|X| \geq |Y|,\ X \not\rightarrow Y}{Y \not\rightarrow X}$

CD-FI2: $\frac{|X| > |Y|}{Y \not\rightarrow X}$

The correctness of these rules is indicated by the fact that for every FD $X \rightarrow Y$ equal X values demand equal Y values. But this implies that there must exist at least as many X values as Y values which can be formulated only with independencies. This fact clearly demonstrates the usefulness of FIs.

Unfortunately, it turned out that testing CDs by our SQL-interface is as expensive as testing FDs[2]. Thus, we check the cardinality of UCDs in a single pass for each attribute and approximate CDs by the following Lemma without a proof:

Lemma 3 *Let $A \in X$ be the attribute with the maximal cardinality and $Y = B_1, \ldots, B_n$. If $|A| \geq (|B_1| \ldots |B_n|)$, then $|X| \geq |Y|$.*

Hence, we initialize our data structures of FDs and FIs with CDs approximated in this manner and the inference rule CD-FI2 only. Since the number of CDs is a combinatorial function of the number of attributes, it is easy to see that the number of CDs grows exponentially w.r.t. the number of attributes. Therefore, the algorithm is in EXPTIME. But, as a matter of fact, this approximation algorithm does not need any resources of the database system. In addition, the worst cases arise only rarely.

Complexity of the System

Our system is in EXPTIME because there exist relations with the number of FDs in a minimal cover growing exponentially w.r.t. the number of attributes. This has been shown by Mannila and Räihä (Mannila & Räihä 1991). As there are relations with the number of FIs growing exponentially the performance cannot be improved by using FIs instead of FDs.

Theorem 3 (Cardinality of the set of FIs) *For each n there exists a relation r which satisfies a minimum cover of FIs with the cardinality $\Omega(2^{n/2})$.*

Proof: see (Bell 1995). □

If the relation of Mannila and Räihä is added to ours, then it is easy to see that relations exist where the sets of FDs and FIs grow exponentially w.r.t. the number of attributes. Again, we argue that our goal, to minimize database access, can be achieved with this system.

Maintenance of FDs

Obviously, the discovered FDs can become invalid, because they only describe the current state of the database. Therefore, the discovered FDs have to be maintained if new tuples are added, old tuples are deleted, or existing tuples are updated.

If maintenance of FDs is seen as revision, it is more suitable to do theory revision than base revision as

[2]We can only count the number of values of single attributes by the SQL statement count.

```
$\Sigma_n := \{\}$
for each $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \in \Sigma$ do
begin
    $b_1, \ldots, b_m, c_1, \ldots, c_l :=$
                select $B_1, \ldots, B_m, C_1, \ldots, C_l$
                from $r$
                where $A_{i_1} = a_{i_1}, \ldots, A_{i_n} = a_{i_n}$
    if $\bigwedge_{i=0,\ldots,m} b_i = d_{n+i}$ then
        $\Sigma_n := \Sigma_n \cup A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$
    else begin
        $\Sigma := \Sigma \setminus A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$
        if there is a minimal $\nu$ with $C_{l_1}, \ldots, C_{l_\nu}$
        and $\bigwedge_{i=1,\ldots,\nu} d_{l_\nu} \notin c_{l_\nu}$ then
        $\Sigma_n := \Sigma_n \cup A_1, \ldots, A_n, C_{l_1}, \ldots, C_{l_\nu}$
                                    $\rightarrow B_1, \ldots, B_m$
    end
end
```

Table 2: Algorithm for Inserting New Tuples

introduced by Gärdenfors (Gärdenfors 1988). In contrast to theory revision, base revision works on the whole consequnce set So it is easy to see, for example, that by adding a new tuple the second FD of the set $\{AB \rightarrow CD, CD \rightarrow EF\}$ may become invalid, but the $AB \rightarrow EF$ remains valid. Therefore, in a first step the minimal set of FDs Σ is transformed into a set Σ_m of FDs. This new set is called *most general* and can be computed with the following algorithm:

```
for each $X \rightarrow Y \in \Sigma$ do
    if $Y = closure(X, \Sigma) \setminus X$ then
        $\Sigma_m := \Sigma_m \cup \{X \rightarrow Y\}$
```

Obviously, the complexity depends on the cardinality of Σ and the closure operation which is mentioned above.

Inserting Tuples

If new tuples are added, FDs may become invalid. Thus, each FD is checked if it is still valid. If not then the FD has to be replaced by a set of FDs which are valid. The algorithm is listed in table 2 and is applied before the tuple is inserted. Let $(d_1, \ldots, d_{n+m+l})$ be the new tuple, r the corresponding relation with the relation scheme $R = (A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_l)$, $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ the selected FD and $C_1, \ldots, C_l$ the remaining attributes. The c_i can consist of values and that we expand the left hand side of each invalid FD by attributes which values are different from the selected ones.

The advantage of this algorithm is that it only one simple query for each new tuple and for each FD is needed. It is easy to see that the removed FDs are no longer valid. For the correctness, we first give the following lemma:

Lemma 4 *Each generated FD is valid.*

Proof: We call the new tuple t_1. We know that $A_1, \ldots, A_n, B_1, \ldots, B_m$ is invalid. Then there must be at least one tuple t_2 so that $t_1[A_1, \ldots, A_n] \overset{n}{=} t_2[A_1, \ldots, A_n]$ and $t_1[B_1, \ldots, B_m] \overset{n}{\neq} t_2[B_1, \ldots, B_m]$. We know by construction that $t_1[C_{l_1}, \ldots, C_{l_v}] \overset{n}{\neq} t_2[C_{l_1}, \ldots, C_{l_v}]$. Hence, $t_1[X, C_{l_1}, \ldots, C_{l_v}] \overset{n}{\neq} t_2[X, C_{l_1}, \ldots, C_{l_v}]$ and $A_1, \ldots, A_n, C_{l_1}, \ldots, C_{l_v} \rightarrow B_1, \ldots, B_m$ is valid. □

Completeness can now be seen by the following lemma:

Lemma 5 *Let Σ be the former set of FDs and Σ_n be the revised set of FDs. r_n is obtained by expanding r by one tuple. Assume that $\Sigma \models X \rightarrow Y$ and $r \models X \rightarrow Y$. If $\Sigma_n \not\models X \rightarrow Y$, then $r_n \not\models X \rightarrow Y$.*

Proof: By correctness we only add valid FDs. By minimality of the added attributes to the LHS of the FD, it is guaranteed that an FD with less attributes at the LHS is not valid. □

We conclude that if our algorithm is applied on Σ and the result is Σ', then $r' \models \Sigma'$ by the lemmas.

Deleting Tuples

Deleting tuples does not affect the old set of FDs, but some new FDs may become valid. Therefore, we have to revise the FDs and add new FDs if necessary. Unfortunately it turned out, that deleting tuples is the same as discovering FDs with a given starting set of FDs. Therefore computation in this case can be as expensive as the discovery process.

Updating Tuples

Normally, updating tuples can be seen as a combination of deleting and inserting tuples. But sometimes we can simplify this process by comparing the old values with the new ones.

Assume that $d = (a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_l)$ is the tuple which will be updated by the values $(a'_1, \ldots, a'_n, b'_1, \ldots, b'_m, c'_1, \ldots, c'_l)$, and the selected FD is $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$.

- If the values of the attributes of the left and the right hand side do not change, then we have nothing to do.
- If $\bigwedge_{i=1,\ldots,m} b_i = b'_i$ and at least one value of A_i, $1 \leq i \leq n$, does not occur in r, then we have only to apply the algorithm for deleting tuples.

Discussion

Our goal has been to minimize database access during the discovery of FDs and the maintenance of the discovered FDs. We have shown, that this can be achieved by the axiomatization of functional dependencies and independencies presented in this paper and the use of cardinality dependencies. The alternative to a complete inference would be a more or less exhaustive test of FDs on the database. Usually, real world databases are very large, the number of tuples is much larger than the number of attributes. The main costs of database management systems are caused by reading from secondary memory. Therefore, a single saved database query makes up for the costs of inferring FDs and FIs. This is true for the maintenance, too.

Basically, our system is to discover candidate keys because these can be used efficiently for optimization techniques. Candidate keys are based on FDs so we have concentrated on only discovery of FDs. In general, algorithms for discovering FDs are in EXPTIME w.r.t. the number of attributes. We have already investigated discovering foreign keys which play also an important role in SQO.

Currently, we are investigating the use of data dependencies in semantic query optimization.

Acknowledgment: I would like to thank Peter Brockhausen, Martin Mühlenbrock, Steffo Weber and Sabine Wohlrab for comments. This work is partly supported by the European Community (ESPRIT Basic Research Action 6020, project Inductive Logic Programming).

References

Bell, S., and Brockhausen, P. 1995. Discovery of constraints and data dependencies in databases (extended abstract). In Lavrac, N., and Wrobel, S., eds., *Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning, 1995)*, Lecture Notes in Artificial Intelligence 914, 267 – 270. Berlin, Heidelberg, New York: Springer Verlag. An extended version is also available as Research Report by the authors.

Bell, S. 1995. Inferring data independencies. Technical Report 16, University Dortmund, Informatik VIII, 44221 Dortmund, Germany.

Chakravarthy, U. S.; Grant, J.; and Minker, J. 1990. Logic–based approach to semantic query optimization. *ACM Transaction on Database Systems* 15(2).

Dehaspe, L.; Laer, W. V.; and Raedt, L. D. 1994. Applications of a logical discovery engine. In Wrobel, S., ed., *Proc. of the Fourth International Workshop on Inductive Logic Programming*, GMD-Studien Nr. 237, 291–304. St. Augustin, Germany: GMD.

Gärdenfors, P. 1988. *Knowledge in Flux — Modeling the Dynamics of Epistemic States.* Cambridge, MA: MIT Press.

Hsu, C.-N., and Knoblock, C. A. 1993. Learning database abstractions for query reformulation. In *Knowledge Discovery in Database, Workshop, AAAI-93.*

Janas, J. M. 1988. Covers for functional independencies. In *Conference of Database Theory.* Springer, Lecture Notes in Computer Science 338.

Jarke, M.; Clifford, J.; and Vassiliou, Y. 1984. An optimizing prolog front–end to a relational query system. *ACM SIGMOD.*

Kanellakis, P.; Cosmadakis, S.; and Vardi, M. 1983. Unary inclusion dependencies have polynomial time inference problems. *Proc. 15th Annual ACM Symposium on Theory of Computation.*

Kanellakis, P. 1990. *Formal Models and Semantics, Handbook of Theoretical Computer Science.* Elsevier. chapter Elements of Relational Database Theory, 12, 1074 – 1156.

King, J. J. 1981. Query optimization by semantic reasoning. Technical Report STAN-CS-81-857, Stanford University.

Maier, D. 1980. Minimum covers in the relational database model. *Journal of the ACM* 27(4):664 – 674.

Mannila, H., and Räihä, K.-J. 1991. *The design of relational databases.* Addison-Wesley.

Paredaens, J.; de Bra, P.; Gyssens, M.; and van Gucht, D. 1989. *The Structure of the Relational Database Model.* Springer Verlag Berlin Heidelberg.

Savnik, I., and Flach, P. 1993. Bottum-up indution of functional dependencies from relations. In Piatetsky-Shapiro, G., ed., *KDD-93: Workshop on Knowledge Discovery in Databases.* AAAI.

Schlimmer, J. C. 1991. Database consistency via inductive learning. In *Eight International Conference on Machine Learning.*

Schlimmer, J. 1993. Using learned dependencies to automatically construct sufficient and sensible editing views. In Piatetsky-Shapiro, G., ed., *KDD-93: Workshop on Knowledge Discovery in Databases.* AAAI.

Siegel, M., and Madnick, S. M. 1991. A metadata approach to resolving semantic conflicts. In *Conference on Very Large Databases.*

Siegel, M.; Sciore, E.; and Salveter, S. 1991. Rule discovery for query optimization. In *Knowledge Discovery in Databases.* Menlo Park: AAAI Press. chapter 24.

Siegel, M. D. 1988. Automatic rule derivation for semantic query optimization. In *Second International Conference on Expert Database Systems.*

Zhang, X., and Ozsoyoglu, Z. M. 1994. Reasoning with implicational and referential constraints in semantic query optimization. In *Workshop on Constraints and Databases, Post-ILPS.*