

Database Reformulation with Integrity Constraints (extended abstract)

Rada Chirkova*
Department of Computer Science
North Carolina State University
chirkova@csc.ncsu.edu

Michael R. Genesereth
Department of Computer Science
Stanford University
genesereth@cs.stanford.edu

Abstract

In this paper we study the problem of reducing the evaluation costs of queries on finite databases in presence of integrity constraints, by designing and materializing views. Given a database schema, a set of queries defined on the schema, a set of integrity constraints, and a storage limit, to find a solution to this problem means to find a set of views that satisfies the storage limit, provides equivalent rewritings of the queries *under the constraints* (this requirement is weaker than equivalence in the absence of constraints), and reduces the total costs of evaluating the queries. This problem, *database reformulation*, is important for many applications, including data warehousing and query optimization. We give complexity results and algorithms for database reformulation in presence of constraints, for conjunctive queries, views, and rewritings and for several types of constraints, including functional and inclusion dependencies. To obtain better complexity results, we introduce an *unchase* technique, which reduces the problem of query equivalence under constraints to equivalence in the absence of constraints *without* increasing query size.

1 Introduction

In many contexts it is beneficial to answer database queries using derived data called *views*. A view is a named query, which can be stored in a

database system as a definition (*virtual view*) or as an answer to the query (*materialized view*). A user query can be answered using views via a new definition that is called a *rewriting* and is built in terms of the views. Using virtual or materialized views in query answering [LMSS95] is relevant in applications in information integration, data warehousing, web-site design, and query optimization. Two main directions in answering queries using views are (1) *feasibility*: to obtain some answer to a given query using given views, as in the information-integration scenario, and (2) *efficiency*: to reduce query-execution time by using the views, as in the query-optimization scenario. Within the efficiency direction, which is our focus in this paper, the objective is typically to use views to obtain *equivalent* query rewritings — that is, definitions that give the exact answer to the query on all databases. Answering queries using views has been explored in depth for relational database systems [Kan90] and for conjunctive queries, which can be defined via positive existential conjunctive formulas of first-order logic [End72]; for a survey of methods for answering queries using views see [Hal01].

In the past few years, significant research efforts have been concentrated on *view selection*, that is, on developing methods for defining and precomputing materialized views to answer predefined queries; existing approaches differ in their main objective (feasibility or efficiency) and in how they explore the search space of views and rewritings for the given queries, typically on finite databases. [CG00] introduced the ap-

*This author's work on this material has been supported by the National Science Foundation under Grant No. 0307072

proach of database reformulation, with an emphasis on efficiency and on *complete* exploration of the search space of efficient rewritings. Formally, starting with a set of finite database relations and a set of queries, the problem is to design a set of views of the database relations that (1) can be materialized under a given restriction (such as a storage limit, i.e., the amount of disk space available for storing the view relations) and, once materialized, (2) can be used by a given evaluation algorithm in answering the queries equivalently and more efficiently than the original relations. The schema consisting of the materialized views is called a *reformulation* of the problem input. A reformulation is *beneficial* (or *optimal*) if it is as efficient as or more efficient than the original (or every other) [re]formulation on all given queries and all databases consistent with the given schema. It has been shown [CG00] that there are reformulation problems for which there are infinitely many beneficial reformulations; at the same time, only finitely many of these reformulations need to be considered since any other reformulation is either larger or less efficient to use. Therefore, it is possible to find an optimal reformulation in finite time.

The results in [CG00] do not take into account integrity constraints, or dependencies, on the base relations in the database. Dependencies are semantically meaningful and syntactically restricted sentences of the predicate calculus that must be satisfied by any “legal” database; examples include functional dependencies and foreign-key constraints [Kan90, AHV95]. The presence of dependencies can increase the set of beneficial reformulations of a database. Consider an example:

EXAMPLE 1.1 Let a query Q be defined on a database with schema $\{S(A, B), T(C, D)\}$ as $q(X, Y) : - s(X, Y), s(X, a), t(Y, a)$.

Consider a view V ,

$v(X, W) : - s(X, a), t(a, W)$.

Query Q — but not view V — *has self-joins*, that is, the definition of Q but not of V has multiple literals with the same relation name. It can be shown [LMSS95] that in the absence of dependencies, V cannot be used to equivalently rewrite

Q . At the same time, suppose the database satisfies a functional dependency σ ,

$\sigma : \forall X, Y, Z (s(X, Y) \wedge s(X, Z) \rightarrow (Y = Z))$.

This dependency means that whenever two tuples in relation S agree on the value of the first attribute A , they also agree on the value of the second attribute B of S .

On all databases satisfying the dependency σ , the query Q can be equivalently rewritten¹ using the view V , as follows:

$q(X, a) : - v(X, a)$.

The reformulation is optimal on all databases satisfying σ , as the materialized view V precomputes an exact answer to Q . \square

In this paper we enhance the results of [CG00] to deal with the additional complexities that arise in presence of dependencies. The problem we consider is as follows: given a set of queries, a set of dependencies, and a storage limit, is it possible to efficiently generate reformulations that satisfy the storage limit and minimize the total costs of evaluating the queries, in the presence of the dependencies. We look at this problem for conjunctive queries, views, and rewritings on finite databases in presence of several types of dependencies, including functional and inclusion dependencies. Our results are applicable in data warehousing and query optimization. Our contributions are as follows:

- we give a new algorithm and tighter complexity results for database reformulation in the absence of dependencies, for queries without self-joins (Section 2.4);
- we give complexity results and algorithms for database reformulation in presence of dependencies, based on the chase technique [AHV95] for incorporating dependencies into query definitions (Section 3);
- we introduce an *unchase* technique for reducing the problem of query equivalence under dependencies to query equivalence in the absence of dependencies, *without* increasing query size (Section 4);

¹We assume set semantics [CV93] for query evaluation.

- we show that we can reduce the complexity of database reformulation *and* cover larger classes of dependencies by basing the reformulation algorithm on the unchase approach (Section 4).

After covering related work in the remainder of this section, we give basic definitions and formal problem statement in Section 2. We then present complexity results and algorithms for database reformulation: Section 3 describes an approach based on chase, and Section 4 discusses our unchase technique. We conclude and discuss future work in Section 5.

Related work

Studies of dependencies have been motivated by the goal of good database schema design; interestingly, they have also contributed to basic research in mathematical logic. The study of dependency theory began with the introduction of functional dependencies in [Cod72]; inclusion dependencies were first identified in [CFP84]. The topic of queries defined over databases that satisfy dependencies was initiated in [ASU79b, ASU79a]. Containment in the presence of inclusion dependencies has been examined in [KCV83, JK84]. For surveys and references on data dependencies, see [FV84, Kan90, AHV95].

An important technique named *chase* grew out of the algorithm of [ABU79] for testing loss-less joins. The chase can be further extended into a semidecision procedure for embedded-dependency implication and an exponential decision procedure for full dependency implication, see [BV84b, BV84a]. In its most general form, chase is similar to resolution with paramodulation. See [Deu02, DLN05] and references therein for applications of chase to answering queries equivalently using views.

Conjunctive queries [CM77, ASU79b, ASU79a] form a large and well-studied class of queries that contains a large proportion of those questions one might wish to ask in practice. When there are no dependencies to consider, or when there are only functional dependencies, it has been shown that the containment, equivalence, and minimization problems are all

NP-complete [CM77]. These results should not be viewed as negative, especially for problems concerned with query optimization, since queries are typically much smaller than the databases on which they are asked, and queries may be applied repeatedly over time [JK84].

References to view selection can be found in [Hal01, CHS02, AC05]. To the best of our knowledge, the results presented here are the first results on view selection in presence of dependencies.

2 Preliminaries

In this section we provide definitions and technical background for our framework, using in part the materials in [Kan90, AHV95].

2.1 Basic definitions

A *relational database* is a finite collection of stored relations. Each relation R is a finite set of tuples, where each tuple is a list of values of the attributes in the *relation schema* of R . We consider select-project-join SQL queries with equality comparisons, a.k.a. *safe conjunctive queries*. A conjunctive query is a rule of the form: $Q : q(\bar{X}) \leftarrow e_1(\bar{X}_1), \dots, e_n(\bar{X}_n)$, where e_1, \dots, e_n are names of database relations and $\bar{X}, \bar{X}_1, \dots, \bar{X}_n$ are vectors of variables. A query Q has *self-joins* if at least two different atoms $e_i(\bar{X}_i), e_j(\bar{X}_j)$ in the body of Q have the same relation name. The variables in \bar{X} are called *head* or *distinguished variables* of Q , whereas the variables in \bar{X}_i are called *body variables* of Q . A query is *safe* if $\bar{X} \subseteq \bigcup_{i=1}^n \bar{X}_i$.

2.2 Dependencies and chase

A *dependency* over a database schema \mathcal{S} is a sentence in some logical formalism over \mathcal{S} . We consider *tuple-generating dependencies (tgds)* and *equality-generating dependencies (egds)* [BV84b]. A tgd is of the form $\forall \bar{x} (\phi(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y}))$, and an egd is of the form $\forall \bar{x} (\phi(\bar{x}) \rightarrow (x_i = x_j))$. Here, $\bar{x} = x_1, \dots, x_k, \bar{y} = y_1, \dots, y_m$, and each of x_i, x_j is an element in \bar{x} . In addition, we consider *consistency constraints* of the form $\forall \bar{x} (\phi(\bar{x}) \rightarrow \text{false})$. In this paper, we consider *conjunctive* egd's, tgd's, and consistency con-

straints, that is, in all the dependencies we consider, $\phi(\bar{x})$ is a conjunction of relational atoms, and $\psi(\bar{x}, \bar{y})$ (in the tgd's) is a single relational atom. We refer to conjunctive egd's as *functional dependencies* (fds), and distinguish between two types of conjunctive tgd's: In *value-preserving tgd's*, \bar{y} in the right-hand side is empty, and in *value-generating tgd's*, \bar{y} contains at least one variable name. In many results in this paper, we focus on a special case of conjunctive tgd's called *inclusion dependencies* (ids), which have just one relational atom in the left-hand side. Inclusion dependencies may be value preserving or value generating. We will use a shorthand notation, in which quantifiers are not used where clear from context. For ids we will also use the notation $r[\bar{x}] \subseteq s[\bar{x}]$, which is equivalent to $\forall \bar{x}, \bar{z} (r(\bar{x}, \bar{z}) \rightarrow \exists \bar{y} s(\bar{x}, \bar{y}))$.

A set Σ of ids is *acyclic* if there is no sequence $r_i[\bar{x}_i] \subseteq s_i[\bar{x}_i]$ ($i \in [1, \dots, n]$) of ids in Σ where for $i \in [1, \dots, n]$, $r_{i+1} = s_i$ for $i \in [1, \dots, n-1]$, and $r_1 = s_n$. A family Σ of dependencies *has acyclic ids* if the set of ids in Σ is acyclic [AHV95]. We define *acyclic tgds* as follows: A set Σ of tgds is *acyclic* if there is no sequence $\sigma_1, \sigma_2, \dots, \sigma_n$ of tgds of Σ , $\sigma_i : r_{i1}(\bar{x}_{i1}) \wedge \dots \wedge r_{ik}(\bar{x}_{ik}) \rightarrow s_i(\bar{y}_i)$ ($i \in [1, \dots, n]$) of tgds in Σ where for $i \in [1, \dots, n]$, the left-hand side of σ_{i+1} includes the relation name for the right-hand side of σ_i , for $i \in [1, \dots, n-1]$, and the left-hand side of σ_1 includes the relation name for the right-hand side of σ_n . A set Σ of n acyclic tgds is *strongly acyclic* if there exists a sequence $\sigma_1, \sigma_2, \dots, \sigma_n$ of tgds of Σ , such that for $i \in [1, \dots, n-1]$ and for all $k > 0$ such that $i+k \leq n$, the right-hand side of σ_{i+k} does not include any relation name in the left-hand side of σ_i . A family Σ of dependencies *has (strongly) acyclic tgds* if the set of tgds in Σ is (strongly) acyclic.

We denote the left-hand side of a dependency (or the body of a query) by A . An *assignment* γ for A is a mapping of the variables appearing in A to constants, and of the constants in A to themselves. Assignments are naturally extended to tuples and atoms; for instance, for a tuple of variables $\bar{s} = (s_1, \dots, s_k)$ we let $\gamma\bar{s}$ denote the tuple $(\gamma(s_1), \dots, \gamma(s_k))$. *Satisfaction* of atoms

by an assignment w.r.t a database is defined as follows: $p_i(\gamma\bar{s})$ is satisfied if the tuple $\gamma\bar{s}$ is in the relation that corresponds to the predicate of p_i . This definition is naturally extended to that of satisfaction of conjunctions of atoms. An *answer to a safe query* Q with head $q(\bar{x})$ and body A on a database \mathcal{D} is the set of all tuples $\gamma(\bar{x})$ such that γ is a satisfying assignment for A on \mathcal{D} .

A database \mathcal{D} *satisfies a set of dependencies* Σ if, for each dependency σ in Σ and for all satisfying assignments γ of the left-hand side of σ w.r.t. \mathcal{D} , σ evaluates to true. (For value-generating tgd's σ , we additionally require that we can extend each γ in such a way that the right-hand side of σ evaluates to true.) For a given set Σ of dependencies and conjunctive queries Q_1 and Q_2 , Q_1 is *contained* in Q_2 under Σ , denoted by $Q_1 \sqsubseteq_{\Sigma} Q_2$, if for any database \mathcal{D} that satisfies Σ , the answer to Q_1 on \mathcal{D} is a subset of the answer to Q_2 on \mathcal{D} . Two queries are *equivalent under* Σ if they are contained in each other under Σ . Query containment and equivalence in the absence of dependencies is defined as above for the case $\Sigma = \phi$ (empty set).

In this paper we use the following results of [CM77] for conjunctive queries. In the absence of dependencies, a query Q_1 is contained in Q_2 if and only if there exists a *containment mapping* from Q_2 to Q_1 , that is, a homomorphism from the variables of Q_2 to the variables and constants of Q_1 , such that (1) each atom in the body of Q_2 is mapped into some atom in the body of Q_1 , and (2) the head of Q_2 is mapped into the head of Q_1 . For a query Q , its *minimized* version is an equivalent query Q' with a minimum number of subgoals, which can be obtained via repeated applications of containment mappings. Two queries are equivalent if and only if their minimized versions are isomorphic.

It is easy to show the following:

Proposition 2.1 *Given a database schema \mathcal{S} , queries Q_1 and Q_2 defined on \mathcal{S} , and a set Σ of dependencies on \mathcal{S} , if Q_1 is contained in Q_2 in the absence of dependencies, $Q_1 \sqsubseteq Q_2$, then Q_1 is contained in Q_2 under Σ , $Q_1 \sqsubseteq_{\Sigma} Q_2$. \square*

The *chase* is a process that, given dependen-

cies Σ , transforms a query Q into a query Q' such that $Q \equiv_{\Sigma} Q'$. A *chase sequence* of a conjunctive query $Q : q : - A$ by a set of dependencies Σ is a (possibly infinite) sequence of conjunctive queries $(q_0, A_0), (q_1, A_1), \dots, (q_i, A_i), \dots$, where $q_0 = q$ and $A_0 = A$, and for each $i \geq 0$, the query (q_{i+1}, A_{i+1}) is the result of applying some dependency in Σ to the query (q_i, A_i) . We can apply a dependency to a query if there is a satisfying assignment γ of the left-hand side of the dependency w.r.t. the body of the query. For fds, the chase rule is to consistently rename query variables according to the equality in the right-hand side of the fd. For inds, the chase rule [JK84] adds to a partial chase result (q_i, A_i) a subgoal that matches the right-hand side $p(\bar{x})$ of the ind, provided no existing subgoal in (q_i, A_i) matches $p(\bar{x})$. This rule is extended to tgds in a natural way. The chase sequence is *terminal* if (1) it is finite, and (2) no dependency in Σ can be applied to the last element in the sequence. The *result* of a terminal chase sequence is its last element (q_n, A_n) , written in query form as $Q' : q_n : - A_n$.

Definition 2.1 (Chase) For a query Q and a set of dependencies Σ , the *chase* of Q by Σ , denoted $chase_{\Sigma}(Q)$, is the result of any terminal chasing sequence of Q by Σ . \square

Given a query Q and dependencies Σ , we compute $chase_{\Sigma}(Q)$ by picking the dependencies in Σ in some arbitrary order and applying them to Q . Importantly, the chase is determined by the *semantics*, rather than the *syntax*, of the dependencies in Σ . Let Σ and Σ' be two sets of dependencies over schema \mathcal{S} . If $\Sigma \equiv \Sigma'^2$, then $chase_{\Sigma}(Q)$ and $chase_{\Sigma'}(Q)$ coincide for any query Q .

The following result has been shown for sets of functional dependencies in [AHV95]; we have extended it to sets of any dependencies considered in this paper.

Theorem 2.1 *Given conjunctive queries Q_1, Q_2 and a set Σ of fds, conjunctive consistency constraints, and conjunctive tgds.*

1. $Q_1 \sqsubseteq_{\Sigma} Q_2$ iff $chase_{\Sigma}(Q_1) \sqsubseteq chase_{\Sigma}(Q_2)$ in the absence of any constraints.

² $\Sigma \equiv \Sigma'$ if $\Sigma \models \Sigma'$ and $\Sigma' \models \Sigma$.

2. $Q_1 \equiv_{\Sigma} Q_2$ iff $chase_{\Sigma}(Q_1) \equiv chase_{\Sigma}(Q_2)$ in the absence of any constraints. \square

2.3 Views and database reformulation

A *view* refers to a named query. A view is said to be *materialized* if its answer is stored in the database. Let \mathcal{V} be a set of views defined on a database schema \mathcal{S} , and \mathcal{D} be a database with schema \mathcal{S} ; by $\mathcal{D}_{\mathcal{V}}$ we denote the database obtained by computing all the view relations in \mathcal{V} on \mathcal{D} . Let Q be a query defined on \mathcal{S} , and \mathcal{V} be a set of views defined on \mathcal{S} . A query R is a *rewriting of Q using \mathcal{V}* if all atoms in the body of R are view predicates defined in \mathcal{V} .

The *expansion R^{exp}* of a rewriting R of a query Q on a set of views \mathcal{V} is obtained from R by replacing all the view atoms in the body of R by their definitions in terms of the base relations. A rewriting R of a query Q on a set of views \mathcal{V} is an *equivalent rewriting* of Q under Σ if for every database \mathcal{D} that satisfies Σ , $Q(\mathcal{D}) = \mathcal{R}(\mathcal{D}_{\mathcal{V}})$.

We consider the following *database-reformulation* problem: Given a set of conjunctive queries \mathcal{Q} on stored relations, a fixed database instance \mathcal{D} that satisfies a set of dependencies Σ , and a storage limit L , we want to find and precompute offline a set of views on the stored relations. A set of views \mathcal{V} is *admissible* for $(\mathcal{Q}, \mathcal{D}, \Sigma, L)$ if (1) \mathcal{V} provides an equivalent rewriting for each query in \mathcal{Q} under Σ , and (2) the total size of the relations for \mathcal{V} on \mathcal{D} does not exceed the storage limit L . (The *size* of a relation is the number of bytes used to store the relation.) Among such admissible sets of views, our goal is to find a *beneficial* (or *optimal*) viewset, that is, a set of views whose use in rewritings of the queries in \mathcal{Q} reduces (minimizes) the sum of evaluation costs of these queries on the database \mathcal{D} satisfying the dependencies Σ . For query-evaluation costs, we consider *size-monotonic* cost models, where (1) query costs are computed using the sizes of the contributing relations, and (2) whenever a relation in a query expression is replaced by another relation of at most the same size, the cost of evaluating the new expression is at most the cost of evaluating the original expression.

All the common cost models in the literature are size-monotonic.

Definition 2.2 (Database reformulation)

For a problem input $\mathcal{I} = (\mathcal{Q}, \mathcal{D}, \Sigma, L)$, a *beneficial (optimal) viewset* is a set of views \mathcal{V} defined on \mathcal{S} , such that: (1) \mathcal{V} is an admissible viewset for \mathcal{I} , and (2) \mathcal{V} reduces (minimizes) the total cost of evaluating the queries in \mathcal{Q} on the database $\mathcal{D}_{\mathcal{V}}$. \square

We consider this problem in relational databases for conjunctive queries, views, and rewritings. We assume that filtering views are not used in query rewritings.³ In some results we additionally assume that input queries do not have self-joins. We use these simplifying assumptions to do an initial study of the structure of the database-reformulation problem under dependencies. It is known that when these assumptions do not hold, the problem has a triply exponential upper bound and a singly exponential lower bound even in the absence of dependencies [CHS02]. The database-reformulation problem is in NP in the absence of dependencies when input queries do not have self-joins and when filtering views are not used [ACGP05].

2.4 The `cgalg` algorithm [CG00]

We now outline an algorithm for generating beneficial reformulations for the case where the set of dependencies Σ is empty and \mathcal{Q} comprises a single query Q [CG00]. For each beneficial reformulation (viewset) \mathcal{V} for a problem input \mathcal{I} , this algorithm generates at least one beneficial reformulation (viewset) \mathcal{V}' that reduces the costs of the input query workload at least as much as \mathcal{V} and satisfies the same storage limit. We say that the algorithm produces the *best* beneficial database reformulations.

Procedure `cgalg`.

Input: query Q , database \mathcal{D} , storage limit L .

Output: R_{opt} , optimal equiv. rewriting of Q on \mathcal{D} .

1 Begin:

2 minimize Q to obtain a query Q' ;

3 set R_{opt} to Q' ;
 4 set the cost C_{opt} of R_{opt} to $C(Q')$;
 5 find all views \mathcal{V} whose body is a subset of subgoals of Q' ;
 6 for each subset \mathcal{W} of \mathcal{V} such that $\Sigma_{\mathcal{W}} \in \mathcal{W}size(\mathcal{W}, \mathcal{D}) \leq L$ do:
 7 begin:
 8 find a rewriting R of Q' using \mathcal{W} ;
 9 construct the expansion R^{exp} of R ;
 10 if there exists a containment mapping from Q' to R^{exp} then:
 11 if the cost $C(R, \mathcal{D}, \mathcal{O})$ of answering Q' on \mathcal{D} using R is less than C_{opt}
 12 then begin:
 13 $R_{opt} := R$;
 14 $C_{opt} := C(R, \mathcal{D}, \mathcal{O})$;
 15 end;
 16 end;
 17 return R_{opt} .
 18 End.

A view-size oracle \mathcal{O} instantaneously gives the size of any relation defined on the database \mathcal{D} ; we assume that for a rewriting R in terms of views and for a fixed size-monotonic cost model for query evaluation, the time required to obtain the cost $C(R, \mathcal{D}, \mathcal{O})$ of evaluating R in terms of the relations for the views on \mathcal{D} is negligible when using the oracle \mathcal{O} . In practice, the view sizes and costs of answering Q on \mathcal{D} using R can be estimated via standard formulas used in query optimizers in database-management systems. It is easy to see how the `cgalg` algorithm can be extended to problem inputs with non-singleton query workloads.⁴

Proposition 2.2 [CG00, ACGP05] *Given $\Sigma = \phi$ and provided that all view atoms in all rewritings have different relation names and that filtering views are not used in query rewritings, the algorithm `cgalg` is sound for problem inputs with workloads of arbitrary conjunctive queries and is complete for problem inputs with workloads of conjunctive queries without self-joins. The de-*

³In an equivalent rewriting R of a query Q , a view V is a *filtering view* if the result of removing the literal for V from R is still an equivalent rewriting of Q .

⁴When queries have no self-joins and each view in \mathcal{V} is used exactly once in the rewriting of exactly one query in \mathcal{Q} [ACGP05], `cgalg` can look for views for each workload query separately even when Σ is not empty.

cision version of the problem of finding optimal reformulations is NP complete. \square

In general, the algorithm is not complete (i.e., is not guaranteed to produce an optimal reformulation) because some optimal rewritings may use self-joins of view literals [CHS02].

Proposition 2.3 *Under the assumptions of Proposition 2.2 and assuming that a view-size oracle \mathcal{O} and a size-monotonic cost model for query evaluation are given, the runtime of `cgalg` is $\Theta(2^m)$, where m is the total number of subgoals of the queries in the input workload \mathcal{Q} . \square*

Intuitively, under the assumptions of Proposition 2.2, `cgalg` will generate all beneficial reformulations if it generates only viewsets that have up to m views [ACGP05]. Note that the step of generating a rewriting given a subset \mathcal{W} of the set \mathcal{V} of views takes constant time in the size of the subset \mathcal{W} [ALU01].

3 Dependencies and Chase

In this section and in Section 4, we consider the database-reformulation problem for workloads of conjunctive queries under a nonempty set of dependencies Σ . In this section our focus is on using chase to extend the `cgalg` algorithm (Section 2.4) to database reformulation in presence of dependencies.

We first observe that the straightforward approach to finding all useful views and rewritings does not really work. Given a query Q and a set of dependencies Σ , we can use Theorem 2.1 to reduce the problem of finding rewriting expansions that are equivalent to Q under Σ to the problem of finding rewriting expansions whose terminal chase result (under Σ) is equivalent, *in the absence of Σ* , to the terminal chase result Q_c of Q under Σ . Even if Q_c is unique and finite, the number of queries that are equivalent to Q_c is infinite [CM77], and the number of *all* beneficial views and rewritings can be infinite [CG00]. In this section we use chase to extend the approach of [CG00] of generating the best (rather than all) beneficial viewsets using the `cgalg` algorithm.

3.1 Consistency constraints

We first obtain that consistency constraints do not generate new views.

Theorem 3.1 *Let \mathcal{I} be a problem input where all dependencies in Σ are consistency constraints. Then an optimal set of views \mathcal{V} for \mathcal{I} can be found by finding an optimal set of views for the problem input that is obtained by removing all dependencies from \mathcal{I} . \square*

A corollary of this result is that if at least one consistency constraint is combined with any number of fds and tgds, then the database-reformulation output is the same as for a problem input where all the consistency constraints are removed.

3.2 Functional dependencies

As we saw in Example 1.1 in Section 1, unlike consistency constraints, fds can generate new beneficial reformulations.

Lemma 3.1 [AHV95] *Let Σ be a set of fds; for any query Q , let $Q' = \text{chase}_\Sigma(Q)$. Then (1) Q' is unique up to variable renamings, and (2) the size of the minimized version of Q' does not exceed the size of the minimized version of Q . \square*

Theorem 3.2 *Algorithm `cgalg`($\{\text{chase}_\Sigma(Q)\}$, \mathcal{D} , L) produces an optimal reformulation of a problem input \mathcal{I} where $\mathcal{Q} = \{Q\}$ and where all dependencies in Σ are fds, provided that all queries in the workload $\{\text{chase}_\Sigma(Q)\}$ have no self-joins. \square*

Note that to produce an optimal reformulation, we only need to consider the *terminal* chase result of each query in the workload \mathcal{Q} . The complexity of `cgalg` here does not exceed the complexity of `cgalg` for the same problem input in the absence of dependencies; note that the original queries may have self-joins (see Example 1.1).

3.3 Conjunctive tgds

We now consider problem inputs whose dependency sets Σ contain acyclic sets of conjunctive tgds. We first consider the case where all tgds are ids.

Proposition 3.1 [AHV95] *Let Q be a query and Σ a set of fds and acyclic ids. Then each chasing sequence of Q by Σ terminates after an exponentially bounded number of steps. \square*

Proposition 3.2 *Let Σ be a set of fds $\Sigma[F]$ and acyclic ids $\Sigma[I]$, $\Sigma[F] \cup \Sigma[I] = \Sigma$. Then for all conjunctive queries Q , $\text{chase}_\Sigma(Q) = \text{chase}_{\Sigma[I]}(\text{chase}_{\Sigma[F]}(Q))$. \square*

This result extends the result of [JK84] for a special class of sets of fds and “key-based” ids; to obtain the extension, we use the observation that the chase rule for ids in [JK84] (which we also use) does not add to the partial chase result $Q_{c,p}$ the right-hand side of a qualifying id if a matching subgoal is already in $Q_{c,p}$. In extending the result to acyclic tgds, the subtlety is that (part of) the left-hand side of a tgd can match the left-hand side of an fd in the same set of dependencies, which would cause Proposition 3.2 to be violated. (For instance, Σ can include a tgd $s(X, Y) \wedge s(X, Z) \rightarrow p(X, Z)$ and an fd $s(X, Y) \wedge s(X, Z) \rightarrow Y = Z$.) We obtain the result of Proposition 3.2 for sets of dependencies Σ that have been preprocessed, by applying each fd in Σ to the left-hand side of each tgd in Σ .

Theorem 3.3 *Algorithm $\text{cgalg}(\{\text{chase}_\Sigma(Q)\}, \mathcal{D}, L)$ is sound for problem inputs \mathcal{I} where $\mathcal{Q} = \{Q\}$ and where Σ is a set of fds and acyclic tgds. The algorithm is complete for such inputs if queries $\text{chase}_\Sigma(Q)$ have no self-joins. \square*

4 Reducing the Complexity by Unchase

In Section 3 we saw that we can obtain the best beneficial reformulations for a workload of conjunctive queries in presence of consistency constraints, fds, and acyclic tgds, either separately or in combination, by using the **cgalg** algorithm on the terminal chase results of the workload queries. At the same time, the restrictions on this approach are rather strong. First, the terminal chase result of each query cannot have self-joins if we want to obtain *optimal* reformulations. Second, as shown in Section 2.4, the complexity of **cgalg** is exponential in the size of the queries to which **cgalg** is applied, that is, to the terminal chase results of the workload queries.

We now give an example where the terminal chase result of a query under acyclic ids (1) has self-joins, and (2) is of size exponential in the

size of the query. Thus, the **cgalg** approach of Section 3 is not guaranteed to produce optimal reformulations in this case, and the cost of using the approach to produce some beneficial reformulations would be prohibitive even for simple queries. However, in this section we give a modified **cgalg** approach that is applicable to the problem input of this example and to other cases, including problem inputs where the terminal chase results of the input queries under the input dependencies are infinite in size.

EXAMPLE 4.1 On a database schema $\mathcal{S} = \{P_1(A_1, B_1), P_2(A_2, B_2), \dots, P_m(A_m, B_m)\}$, consider a query Q with a single subgoal p_1 : $q(X, Y) : - p_1(X, Y)$.

Suppose the database schema \mathcal{S} satisfies a set Σ of acyclic ids of the following form:

$$\sigma_{i,j}^{(1)} : p_i(X, Y) \rightarrow p_j(Z, X)$$

$$\sigma_{i,j}^{(2)} : p_i(X, Y) \rightarrow p_j(Y, W)$$

Σ has one id $\sigma_{i,j}^{(1)}$ and one id $\sigma_{i,j}^{(2)}$ for each pair (i, j) , where $i \in \{1, \dots, m-1\}$ and $j \in \{i+1, \dots, m\}$ ($i < j$ in each pair). Thus, the number of dependencies in Σ is quadratic in m .

We show one partial chase result of the query Q under dependencies Σ , for $m \geq 2$:

$$q'(X, Y) : - p_1(X, Y), p_2(Z_1, X), p_2(Y, Z_2).$$

This query Q' is the result of applying to Q dependencies $\sigma_{1,2}^{(1)}$ and $\sigma_{1,2}^{(2)}$.

For the terminal result Q_c of chasing the query Q under the ids Σ , we can show that the size of Q_c is exponential in the size of Q and Σ . \square

For the problem input in this example, the cost of using **cgalg** of Section 3 is doubly exponential in the size of the query Q , and the problem of finding beneficial reformulations has an exponential-size lower bound, just because we need to output views that cover all the subgoals of this exponential-size terminal chase result. Thus, the terminal chase result of a query under acyclic ids can have an exponential number of views even for (1) nonfiltering views only, and (2) no self-joins in input queries (cf. [ACGP05]).

4.1 Unchase for ids and tgds

The idea we outline in this section is to apply our reformulation algorithm to those versions of the

input queries that have all “derived” subgoals removed. Thus, our approach is to (1) apply “unchase” to all the input queries under the input dependencies, and then to (2) apply `cgalg` to the results of the unchase.

We first define unchase for sets of ids only: Given a finite-size query Q and an id σ , an *unchase step* on (Q, σ) is to remove from Q a subgoal s that is the image, under some homomorphism μ , of the right-hand side r of σ , provided that two conditions are satisfied. First, the homomorphism μ can be extended to map the left-hand side of σ into some subgoal of Q other than s . Second, for each *free* argument Y of r , $\mu(Y)$ in s (1) is a variable rather than a constant, (2) is a nondistinguished variable of Q , and (3) does not occur in any subgoal of Q except s . For instance, if we apply the id $\sigma_{1,2}^{(1)} : p_1(X, Y) \rightarrow p_2(Z, X)$ to query Q' in Example 4.1, we will obtain a query $q''(X, Y) : - p_1(X, Y), p_2(Y, Z_2)$.

For a query Q and for a set of ids Σ , we denote by $Q_{u,\Sigma}$ the terminal unchase result of Q under Σ . Note that unchase under ids terminates in finite time, as each successful unchase step removes a subgoal from the current partial unchase result. We obtain the following uniqueness result for unchase under ids:

Lemma 4.1 *For a conjunctive query Q , for a set of dependencies Σ that has ids only, and for any finite-size (either partial or terminal) chase result Q' of Q under Σ , $Q_{u,\Sigma}$ is equivalent to $Q'_{u,\Sigma}$ in the absence of dependencies.* \square

It follows [CM77] that the result of minimizing $Q_{u,\Sigma}$ is isomorphic to the result of minimizing $Q'_{u,\Sigma}$. Note that in Lemma 4.1 we do not require id acyclicity, and thus the result applies to problem inputs with sets of cyclic ids, such as $\{p(X, Y) \rightarrow p(Y, Z)\}$. We have also extended the result of Lemma 4.1 to sets of strongly acyclic tgds; the unchase rule for tgds is analogous to that for ids. (We require strong acyclicity in the proof to ensure that all tgds can be applied in the unchase process.)

4.2 Unchase in presence of fds

Using Lemma 4.1, we can show that `cgalg` can be applied to the problem input of Example 4.1

to obtain an optimal reformulation from just the terminal unchase result (which is Q itself) of the query Q under the set Σ of ids. However, we can extend the unchase/`cgalg` approach to combinations of ids (or of strongly acyclic tgds) with fds. We first note that if we try to unchase a query using fds only, the unchase process will not terminate in finite time:

EXAMPLE 4.2 For a query

$$q(X, Y) : - p(X, Y).$$

and for a set of dependencies Σ with a single fd, $\Sigma = \{\sigma : p(X, Y) \wedge p(X, Z) \rightarrow Y = Z\}$, an unchase step “add to Q a subgoal p with a fresh variable for the second argument” can be applied infinitely many times. This query Q' is a partial unchase result after two steps:

$$q'(X, Y) : - p(X, Y), p(X, Z_1), p(X, Z_2). \quad \square$$

At the same time, we can guarantee unchase termination and “good” properties of the `cgalg` approach if we incorporate fds into unchase as follows: (1) An unchase step for fds is the same as a “regular” chase step on fds, see Section 2.2. (2) A query is unchased in presence of fds combined with ids (tgds) by applying all the ids (tgds) before all the fds. The complexity of unchase under ids only is $m^3|\Sigma|$, where m is the total number of subgoals in the query workload; the complexity of unchase under ids and fds is $m^4|\Sigma|$.

Proposition 4.1 *For a conjunctive query Q , for a set of dependencies Σ that has fds either alone or in combination with ids or strongly acyclic tgds, and for any finite-size (either partial or terminal) chase result Q' of Q under Σ , $Q_{u,\Sigma}$ is equivalent to $Q'_{u,\Sigma}$ in the absence of dependencies.* \square

To prove this result, we apply and extend the id/fd separability result of [JK84] that says that $\text{chase}_{\Sigma[I+F]}(Q) \equiv \text{chase}_{\Sigma[I]}(\text{chase}_{\Sigma[F]}(Q))$ (for the notation, see Proposition 3.2).

This result is obtained using Proposition 4.1:

Theorem 4.1 *For any two conjunctive queries Q_1 and Q_2 and for a set of dependencies Σ that satisfies the conditions of Proposition 4.1, $Q_1 \equiv_{\Sigma} Q_2$ if and only if $Q_{1,u,\Sigma}$ is equivalent to $Q_{2,u,\Sigma}$ in the absence of dependencies.* \square

To obtain beneficial reformulations for a problem input \mathcal{I} , we apply `cgalg` on the terminal results of unchasing the workload queries in \mathcal{I} under the set of dependencies in \mathcal{I} .

Theorem 4.2 `cgalg`($\{Q_{u,\Sigma}\}, \mathcal{D}, L$) is sound for problem inputs \mathcal{I} where the workload $\mathcal{Q} = \{Q\}$ has conjunctive queries only and such that Σ satisfies the conditions of Proposition 4.1. The algorithm is complete for such problem inputs provided the queries $Q_{u,\Sigma}$ have no self-joins. \square

By definition of the unchase process, the complexity of `cgalg` in this case is $O(2^m)$, where m is the total number of subgoals in the workload queries in the problem input \mathcal{I} .

Theorem 4.3 For problem inputs \mathcal{I} that satisfy the conditions of Theorem 4.2, the decision version of the problem of generating optimal reformulations is in NP, provided that the queries $\text{unchase}_\Sigma(Q)$ have no self-joins. \square

It is remarkable that, given a problem input \mathcal{I} and the rewritings produced by `cgalg` on the terminal results of unchasing the queries in \mathcal{I} using the dependencies in \mathcal{I} , to show the equivalence of the original workload queries to the rewritings, we *do not need* to unchase the expansions of the rewritings. (Note that one needs to apply *chase* to discover rewritings that are equivalent to queries under dependencies; see, e.g., [DLN05]. We can show that if we used the approach described in Section 3, we *would* need to chase the rewriting expansions to show the equivalence of the rewritings to the original queries.)

Theorem 4.4 For problem inputs \mathcal{I} that satisfy the conditions of Theorem 4.2, let R be a reformulation of some query Q in \mathcal{I} , such that R is returned by `cgalg`($\{Q_{u,\Sigma}\}, \mathcal{D}, L$). Suppose $R^{\text{exp}} \equiv Q_{u,\Sigma}$ in the absence of dependencies. Then $R_{u,\Sigma}^{\text{exp}} \equiv Q_{u,\Sigma}$ in the absence of dependencies. \square

5 Conclusions; Future Work

We have presented complexity results and `cgalg` algorithms for database reformulation in presence of dependencies. Our results apply to conjunctive queries and to the types of dependencies

that include commonly used functional dependencies, inclusion dependencies, and foreign-key constraints. We argued that to generate beneficial reformulations, one can use the chase technique for incorporating dependencies into query definitions. At the same time, we showed that we can reduce the complexity of database reformulation and cover larger classes of dependencies by incorporating into the reformulation algorithm our unchase approach; the idea of unchase is to remove from a query all “derived” subgoals that would be introduced by chase.

The unchase/`cgalg` approach can be extended to workloads of queries with self-joins, at the expense of an increase in runtime complexity (cf. [CHS02, ACGP05]). We are currently working on extending the approach to database reformulation for queries with aggregation. Another direction of our ongoing and future work is designing efficient algorithms for database reformulation for common classes of queries and dependencies. Besides database reformulation, the unchase approach can be used in answering queries using views, as it reduces the problem of checking query containment (equivalence) in presence of dependencies to the problem of containment (equivalence) checking in the absence of dependencies, *without* increasing query size. Note that unchase, unlike chase, can be used in presence of cyclic inclusion dependencies. Exploring unchase for answering queries using views is another direction of our future work.

References

- [ABU79] A.V. Aho, C. Beeri, and J.D. Ullman. The theory of joins in relational databases. *ACM TODS*, 4(3):297–314, 1979.
- [AC05] F. Afrati and R. Chirkova. Selecting and using views to compute aggregate queries. In *Proc. ICDT*, 2005.
- [ACGP05] F. Afrati, R. Chirkova, M. Gergatsoulis, and V. Pavlaki. Designing views to efficiently answer *real* SQL queries. In *Proc. SARA*, 2005.

- [AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.
- [ALU01] F. Afrati, C. Li, and J.D. Ullman. Generating efficient plans for queries using views. In *Proc. ACM SIGMOD*, 2001.
- [ASU79a] A.V. Aho, Y. Sagiv, and J.D. Ullman. Efficient optimization of a class of relational expressions. *ACM TODS*, 4(4):435–454, 1979.
- [ASU79b] A.V. Aho, Y. Sagiv, and J.D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.
- [BV84a] C. Beeri and M.Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. Comput.*, 13(1):76–98, 1984.
- [BV84b] C. Beeri and M.Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.
- [CFP84] M.A. Casanova, R. Fagin, and C.H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *JCSS*, 28(1):29–59, 1984.
- [CG00] R. Chirkova and M.R. Genesereth. Linearly bounded reformulations of conjunctive databases. In *Proc. CL*, pages 987–1001, 2000.
- [CHS02] R. Chirkova, A.Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *VLDBJ*, 11(3):216–237, 2002.
- [CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. ACM STOC*, pages 77–90, 1977.
- [Cod72] E.F. Codd. Further normalization of the data base relational model. In R. Rustin, editor, *Data Base Systems*, pages 33–64. Prentice Hall, 1972.
- [CV93] S. Chaudhuri and M.Y. Vardi. Optimization of *real* conjunctive queries. In *Proc. PODS*, pages 59–70, 1993.
- [Deu02] Alin Deutsch. *XML Query Reformulation over Mixed and Redundant Storage*. PhD thesis, University of Pennsylvania, 2002.
- [DLN05] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. In *ICDT*, 2005.
- [End72] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.
- [FV84] R. Fagin and M.Y. Vardi. The theory of data dependencies - an overview. In *ICALP*, pages 1–22, 1984.
- [Hal01] A.Y. Halevy. Answering queries using views: A survey. *VLDBJ*, 10(4):270–294, 2001.
- [JK84] D.S. Johnson and A.C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *JCSS*, 28(1):167–189, 1984.
- [Kan90] P.C. Kanellakis. Elements of relational database theory. In J. van Leeuwen, editor, *Handbook of Theor. CS, Volume B: Formal Models and Semantics*, pages 1073–1156. Elsevier and MIT Press, 1990.
- [KCV83] P.C. Kanellakis, S.S. Cosmadakis, and M.Y. Vardi. Unary inclusion dependencies have polynomial time inference problems. In *STOC*, pages 264–277, 1983.
- [LMSS95] A. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. PODS*, pages 95–104, 1995.