Xie, Dong; Chen, Xinbo; Zhu, Yan
Tackling polytype queries in inconsistent databases: theory and algorithm
Journal of Software, 2012; 7(8):1861-1866

Tackling Polytype Queries in Inconsistent Databases: Theory and Algorithm

Dong Xie
Department of Computer Science and Technology, Hunan University of Humanities, Science and Technology, Loudi, 417000, China
School of Computer Science, The University of Adelaide, SA 5005, Australia
Email: dong.xie@adelaide.edu.au

Xinbo Chen,Yan Zhu
Department of Electronics and Information Engineering, Loudi Vocational and Technical College, Loudi, 417000, China
Email: {cxbo, ldzyjsj}@163.com

***Abstract*—To expand query types under a set of integrity constraints for obtaining consistent answers over inconsistent databases, a computational theory is proposed based on first-order logic. According to directed join graphs of queries and their join completeness, computational complexities of CQA are PTIME if query types are key-key, nonkey-key, incomplete key-key with acyclic join. This paper presents several algorithms to tackle a large and practical class of queries, which can obtain the rewritten queries for computing consistent answers. For a rewritable initial query, a consistent identification statement is constructed based on the join graph by recursive computation; and the statement combines with the initial query to construct a new first-order rewritten query for computing consistent answers. To acyclic self-join queries, the recursive rewriting algorithm cannot eliminate inconsistent tuples, so the initial query combines with the statement that eliminates them.**

***Index Terms*—relational database, inconsistent data; consistent query answer, first-order logic, query rewriting**

I. INTRODUCTION

Integrity constraints (ICs) effectively enable data consistency and validity to conform to the rules of entities in the real-world. The current commercial DBMSes focus on a series of ICs to ensure every database is consistent. However, an entity of the real world frequently corresponds to inconsistent data w.r.t a given set of ICs while data are integrated from different data sources[1,2]. For example, a supplier *S* has two conflict balances: 10 and 50. Give a query for returning suppliers whose balance is more than 20. A common query should return *S*. However, since the balance of *S* may be 10, so the query result is incorrect.

It is difficult or undesirable to repair the database in order to restore consistency. The process may be too expensive, and useful data may be lost. One strategy for managing inconsistent databases is data cleaning[3], which identifies and corrects data errors. However, these techniques are semi-automatic and infeasible for some applications such as a user wants to adopt different cleaning strategies or need to retain all inconsistent data. The trend toward autonomous computing is making the need to manage inconsistent data more acute. As a result, a static approach w.r.t a fixed set of constraints may not be appropriate.

An alternative approach is to employ CQA [1] to resolve inconsistencies at query time. CQA is the problem of retrieving "consistent" answers over inconsistent databases w.r.t a set of ICs. A first-order query rewriting algorithm [1] is only employs conjunctive queries and binary constraints without quantifiers, some tractable queries could not be treated by query rewriting.

Conjunctive queries with quantifiers are based on join graph[4]. Every relation in the query is denoted as a node, an arc from a node to another node if an existential shared variable occurs in a non-key position in a node and occurs also in a key position in another node. Query classes have not repeated relation symbols, and every join condition involves the entire key of at least one relation[5,6]; they address first-order expressibility of acyclic queries without self-join in which no relation name occurs more than once. Since cycles are rare in queries, acyclic queries are very common. Most of queries arising in practice are in the class. In fact, 20 out of 22 queries in the TPC-H decision support benchmark[7] are in the class. As a further result, a dichotomy for a subclass of the class is following: the problem of computing the consistent answers is NP-COMPLETE for not every query in that class whose join graph is a forest [4].

Ref. [8] presents a query answering process based on conflict graph for dealing with denial constraints. It constitutes a compact, space-efficient representation of all repairs of a given database instance, the repairs correspond to maximal independent sets of the graph; vertices of the conflict graph are tuples in the database, an edge connects two vertices if they violate together an IC. Ref. [9] can handle arbitrary relational calculus queries and binary universal constraints. There is a one-to-one correspondence between database repairs and stable

doi:10.4304/jsw.7.8.1861-1866

models of the logic programs. This approach can handle all first-order queries and a much wider class of ICs than the query rewriting technique. However, in the presence of one Functional dependencie(FD) there may be exponentially repairs, the two methods are impractical to compute consistent answers of the bigger database efficiently.

To our best knowledge, no prior work has studied in the context of join completeness of polytype queries over inconsistent databases. Our main works are the following:

(a) This paper proposes the inconsistent data management theory based on the first-order logic and the directed join graph for identifying intractable and tractable queries by analyzing join completeness of polytype queries. Computational complexities of CQA are PTIME while the query join types are key-key, nonkey-key, incompletekey-key and their join graphs are acyclic correspond to queries.

(b) This paper presents several query rewriting algorithms for a large number of practical tractable queries, it first judges whether the queries are rewritable, a rewritable query will be constructed as a new rewritten query, then presents the query rewriting algorithm by recursive computation for subnodes according to the join graph.

II. THEORY

A. First-Order Logic

First-order logic supports a uniform language, which connects various signs to denote a logic deduction as constraints according to certain syntaxes[3]. CQA concentrates on relational databases. A fixed relational schema $S = (U, R, B)$ determines a first-order predicate logic $L(S)$, where U is an infinite database domain, R is a set of database predicates, and B is a set of built-in predicates. Database instances are first-order structures over $L(S)$. We assume relation symbols by $S_1,...,S_m$, tuples of variables and constants by $x, y, ...$, atomic formulas by $A_1,...,A_n$, and quantifier-free formulas by v (contain only built-in predicates). Basic classes of ICs are the following: (a) Universal IC: $\forall A_1 \vee \cdots \vee A_n \vee v$ (binary if n=2); (b)Denial constraint: $\forall \neg A_1 \vee \cdots \vee \neg A_n \vee v$; (c)FD: $\forall x, y, y'$:$(\neg S(\underline{x}, y) \vee \neg S(\underline{x}, y') \vee y = y')$. FD is $X \rightarrow Y$ where X is a set of attributes of S corresponding to x and Y a set of attributes of S corresponding to y; (d) Inclusion dependency: $\forall x,y \exists z$:$(\neg S_1(\underline{x},y) \vee S_1(\underline{y},z))$.

A query $Q(x_1,...,x_n)$ is a first-order form of $L(S)$, where $x_1,...,x_n$ are free variables and $n \geq 0$, $Rep(D,IC)$ is an instance that meets ICs IC. If every $D' \in Rep(D,IC)$:$D' \models Q(t')$, the tuple set $t'=(t_1,...,t_n)$ is a consistent result($x_1,...,x_n$ obtain $t_1,...,t_n$ respectively). If $n=0$, Q is a Boolean query. To every $D' \in Rep(D, IC)$, if $D' \models Q(t')$, Q is true, otherwise it is false.

To indicate key constraints easily, the first attribute of a relation is underlined. For example, $q=\exists x,y,z$:$R_1(\underline{x},y) \wedge R_2(\underline{y},z)$ expresses the key attributes of R_1 and R_2 are x and y respectively. This work focuses on conjunctive queries, which may be expressed as $q(z_1,\ldots,z_k)= \exists\ w_1,\ldots,w_m$:$R_1(\underline{x}_1,y_1) \wedge \ldots \wedge R_n(\underline{x}_n,y_n)$. x indicates a constant or a variable in the key position, y indicates a constant or a variable in the non-key position.Where, $w_1,\ldots,w_m$, are variables in q, $z_1,\ldots,z_k$ are free variables. If w appears in R_i ($\underline{x}_i$, y_i) and R_j ($\underline{x}_j$, y_j) while $i \neq j$, it is a join. If w appears in x_i and x_j, it is a key-key join; if w appears in y_i and y_j, it is a nonkey-nonkey join; if w appears in x_i and y_j or w appears in x_j and y_i,it is a nonkey-key join.

B. Repair

Definition 1. Inconsistent database (IDB) [1]. Given an instance I of a database schema R and a set of ICs IC, we say that I is consistent if $I \models IC$ in the standard model-theoretic sense; inconsistent otherwise.

Definition 2. Distance[1].The distance between two database instances $I1$ and $I2$ is their symmetric difference $\Delta(I1, I2) = (I1 - I2) \cup (I2 - I1)$.

Definition 3. Repair [1]. Given a set of ICs IC and database instances I and I', we say that I' is a repair of I w.r.t IC if $I' \mid\models IC$ and there is no instance I'' such that $I'' \models$ IC and $\Delta((I, I'') \subset \Delta(I, I')$.

I' is a subset of I w.r.t IC, I' is the minimal different for I. Rep (I, IC) indicates I is a repair w.r.t IC. Repairs are not unique, every repair is a subset of I and corresponds with a possible cleaned consistent database.

Difination 4.Consistent query answers (CQA) [1]. A tuple t is a consistent answer to a query $Q(x)$ in a database instance I w.r.t. a set of ICs IC iff t is an answer to the query $Q(x)$ in every repair I' of I w.r.t. IC. We can define *true* being a consistent answer to a Boolean query in a similar way.

Example 1. Assume that a relation schema R ($\underline{name}$,age), and an inconsistent instance I = {R(*Tom,* 20),R(*Tom,* 25), R(*Mary,* 30)}. There are two repairs: I_1={(*Tom,*20), (*Mary,*30)} and I_2={(*Tom,*25), (*Mary,*30)}. All repairs have a minimal distance to I, {(*Tom,*25)} and {(*Mary,*30)} are not repairs because their distances w.r.t I is not minimal under set inclusion. The minimality condition for the repairs is crucial in the definition. Otherwise, the empty set would trivially be a repair of every instance. For example, let $q_1(e)=\exists age$: R(*name,age*). The consistent answers for $q1$ on I are the tuples (*Tom*) and (*Mary*). Let q_2 (*name, age*) = R (*name, age*). The only consistent answer for q_2 on I is (*Mary,* 30). Notice that the tuples (*Tom,* 20) and (*Tom,* 25) are not consistent answers. The reason is that neither of them is present in both repairs.

In fact, it is easy to see that there may be exponentially many repairs in the size of the database. Ideally, querying the given inconsistent databases should obtain consistent answers by, avoiding the explicit computation of repairs and filtering candidate answers. The input to the CQA problem is: a schema R, a set of ICs IC, and a database instance I over R, I might violate IC. A repair I' of I is an instance of R such that I' satisfies Σ, and I' differs minimally from I. A tuple t is said to be a consistent answer for a query q on I if $I' \models q[t]$, for every repair I' of I. Under this definition, repairs need not be unique.

Intuitively, each repair may be a possible consistent database.

C. Directed Join Graph

In order to define such conditions precisely, we will state them in terms of what we call the *directed join graph* of the query. In the graph, nodes express relations of the query, arcs express attributes of the query.

Definition 5. Let Q is a *SPJ* query, R_i and R_j are two relations in Q. The graph G of Q is a directed join graph, if G meets a certain following conditions, we say that the query is polytype. (a)the vertices of G are the relations used in Q; (b)all joins involve the key of at least one relation; (c)there is an arc from R_i to R_j if an attribute of R_i is equated with the key attribute of R_j; (d)G is a tree; (e)a relation appears in the FROM clause at most once;(f)the key of the relation at the root of G appears in the SELECT clause.

The first condition expresses that every relation is used at most once in a query. The second one expresses that every join involves the key of at least one relation. The third one expresses that a non-key attribute of a relation is equated with a key attribute of another one. The fourth condition expresses that self-join does not exist. The fifth condition expresses that a key of a relation joins non-key attributes of the other two relations do not exist.

The most form of joins is from a non-key attribute of a relation to the key of another one. Furthermore, such joins typically involve the primary key of the relation. Finally, cycles are rarely present in the queries used in practice, which do not have repeated relation symbols.

If Q is a conjuctive query and its join graph is directed, Q is rewritable. We divide queries into several types as the following:

$q_1 = \square x,z,y$: $R_1(\underline{x}, y) \wedge R_2(\underline{z}, y)$,$q_1$ is a nonkey-nonkey join;

$q_2 = \square x,y,z,w{:}R1(\underline{x},y) \wedge R2(\underline{z},y) \wedge R3(\underline{y},w)$,$q_2$ is a nonkey-key join, and the number of nodes with in-degree 0 is 2, so it is not a root tree ;

$q_3 = \square x,y,z{:}R1(\underline{z},\underline{x},y) \wedge R_2(\underline{y},x)$, q_3 is a nonkey-key join with a cyclic, and the join of x over the key attribute of $R1$ is not complete;

$q_4 = \square x,y,z{:}R1(\underline{x},z) \wedge R2(\underline{y},x)$,$q_4$ is a nonkey-key join;

$q_5 = \square x,y{:}R1(\underline{x},y) \wedge R2(\underline{y},x)$,$q_5$ is a nonkey-key join with a cyclic between repeated relations;

$q_6 = \square x{:}R1(\underline{x}) \wedge R2(\underline{x})$,$q_6$ is a key-key join;

$q_7 = \square x,y,z,w{:}R1(\underline{x},y) \wedge R2(\underline{y},z) \wedge R3(\underline{z},w) \wedge R4(\underline{z},c) \wedge R5(\underline{y},c)$,$q_7$ is a nonkey-key join.

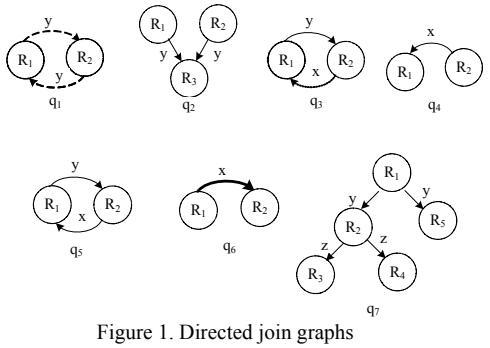

Figure 1. Directed join graphs

D. Intractable Query

Definition 6.Assume a conjunctive query $Q{:}R_i(\underline{x_i},y_i)$ and $R_j(\underline{x_j},y_j)$, if every variable of x_j appear in y_i or x_i, the join is complete; if every variable of x_j does not appear in y_i or x_i, the join is not complete.

We don't present a kind of query: every variable of x_j does not appear in $x_i(i=j)$, it is not complete nonkey-key join with repeated relations. We should prove the type of query is impossible.

Theorem 1. Assume a conjunctive query $Q{:}R_i(\underline{x_i},y_i)$ and $R_j(\underline{x_j},y_j)$, if every variable of x_j does not appear in $x_i(i=j)$, the type of query is impossible.

Proof: Assume the type of query is possible, the number of attributes of x_j is less than x_i . Since $i=j$, $x_i=x_j$, the number of attributes of x_j is equal to x_i. Contradiction.

Theorem 2[10]. To a given conjunctive query with multi-relations and key constraints, if its join types are nonkey-nonkey, computing CQA is NP-COMPLETE problem.

However, the type of conjunctive queries based on key constraint is nonkey-nonkey with single relation, it generates cycles in its join graph. The type of queries does not belong to C_{tree} type[4]. To C_{tree} type, if relax join constraints (e.g., nonkey-key join is not complete), this raises NP-COMPLETE problem. To the type of computable conjunctive queries, iff join graphs of conjunctive queries are not cyclic. However, a large number of practical conjunctive queries are rewritable for first-order logic, computing their CQA are tractable. We should prove that nonkey-nonkey joins with multi-relations are also NP-COMPLETE problem.

Theorem 3. To a given conjunctive query with multi-relations and key constraints, if its join types are nonkey-nonkey, computing CQA is NP-COMPLETE problem.

Proof: This proof may simplify by MONOTONE 3-SAT. Set$\beta=\delta_1 \wedge \ldots \wedge \delta_m \wedge \Psi_{m+1} \wedge \ldots \wedge \Psi_p$ be a conjunctive query, δi is positive, Ψi is negative. Assume two binary relations $R1$ and $R2$, the first attribute of the relation is the key attribute. We create an instance I, if variable v appears in δ_i, $I(R1)$ includes tuple (i,v); if variable v appears in Ψ_i, $I(R2)$ includes tuple(i,v); so $Q \equiv \exists x, y, z{:}\ R1(\underline{x}, y) \wedge R2(\underline{z}, y)$.Iff exist a repair of I as $I^* \not\models Q$, which is satisfied with β.

Theorem 4. To a given conjunctive queries with multi-relations and key constraint, if its join types are nonkey-nonkey and thieir join graphs exist cycles, computing CQA is NP-COMPLETE problem.

Proof: This proof may simplify by MONOTONE 3-SAT. Set $\beta=\delta_1 \wedge \ldots \wedge \delta_m \wedge \Psi_{m+1} \wedge \ldots \wedge \Psi_p$ be a conjunctive query, δ_i is positive, Ψi is negative. Assume two binary relations $R1$and $R2$, the first attribute of the relation is the key attribute. We create an instance I, if variable v appears in δ_i,$I(R1)$ includes tuple (i,v); if variable v appears in Ψ_i, $I(R2)$ includes tuple(i,v); so $Q \equiv \exists x, y{:}R1(\underline{x}, y) \wedge R2(\underline{y}, \mathrm{x})$. Iff exist a REPAIR of I as $I^* \not\models$Q, which is satisfied with β.

Theorem 5[10]. Assume two queryies Q: $\exists x,y,m,z:R1(\underline{x},y) \wedge R2(\underline{m},\underline{y},z)$ and Q^*: $\exists x,x',y:R1(\underline{x},y) \wedge R2(\underline{x'},y)$. It is a PTIME reduction from Q^* to Q for computing, so it is a PTIME reduction from join among part key attributes to join among nonkeys for computing CQA.

Theorem 6. To a given conjunctive query and key constraints, if its nonkey-key join types are not complete, computing CQA is NP-COMPLETE problem.

Proof: According to Theorem 5, it is a PTIME reduction from join among part key attributes to join among nonkeys for computing CQA, so it is a PTIME reduction from join with nonkey-nonkey attributes to incomplete join with nonkey-key for computing CQA. Since the nonkey-nonkey join for computing CQA is NP-COMPLETE problem, the nonkey-key join for computing CQA is NP-COMPLETE problem.

Theorem 7. To a given conjunctive query and key constraints, if its key-key join types without repeated relations are complete incomplete , computing CQA is NP-COMPLETE problem.

Proof:According to Theorem 5, it is a PTIME reduction from join among part key attributes to join among nonkeys for computing CQA, so it is a PTIME reduction from join with nonkey-nonkey attributes to complete incomplete join with key-key for computing CQA. Since the nonkey-nonkey join for computing CQA is NP-COMPLETE problem, the incomplete join with nonkey-key for computing CQA is NP-COMPLETE problem.

E. Tractable Query

Theorem 8[10]. To a given conjunctive query and key constraints, if its joins are key-key or its nonkey-key joins are complete while the join graphs corresponds to queries are not cyclic, computing CQA is NP-COMPLETE problem.

Theorem 9. To a given conjunctive query and key constraints, if its key-key joins are not complete, computing CQA is PTIME.

Proof:According to Theorem 5, it is a PTIME reduction from join among part key attributes to join among nonkeys for computing CQA, so it is a PTIME reduction from join with nonkey-key to incomplete join with key-key for computing CQA. According to Theorem 8, computing CQA for nonkey-key is PTIME, so computing CQA for incomplete join with key-key is PTIME.

According to Theorem 8 and Theorem 9, computing CQA for q_4, q_6 and q_7 are PTIME. According to Theorem 2,Theorem 3,Theorem 4,Theorem 6 and Theorem 7, computing CQA for q_1,q_2,q_3 and q_5 are NP-COMPLETE problem.

The second query exists a nonkey-key join, but it has two nodes whose in-degrees are 0. In fact, the join between *R1* and *R2* is nonkey-nonkey join, these nodes are greater than 1 while their in-degrees are 0, the nonkey-key join is a subclass of common nonkey-key joins. As a result, we classify three types as nonkey-nonkey joins such as the nonkey-key join whose nodes are greater than 1 while their in-degrees are 0, the incomplete nonkey-key join and the complete incomplete key-key join. According to this class, q_1, q_2 and q_3 are the nonkey-nonkey join.

A great number of intractable query types exist, but these intractable queries are rarely exist. Since cycles rarely appear in pratical queries, q_{15} is prevailing. More queries join between a nonkey attribute of a relation and a key attribute of another one, the join with nonkey-key is complete.

Query rewriting employs first-order rewriting method to rewrite a great number of tractable querie, rewritten queries are also first-order, they may be expressed by SQL. An original query and its rewritten query are same type, computing CQA can be reused by same database query engine, this economizes functions of current DBMSes and need not preprocessing and reprocessing of procedures. Since rewritten queries should be obtained by PTIME way with independent data, the method is PTIME on computation complexity.

Definition 7. Assume R is a schema, IC is a set of ICs,q is a query over R. To every instance I of R,t is consistent results w.r.t. IC, if a query Q exist and $I \models Q(t) \equiv t \in CQA(q,I,IC)$, Q is a first-order query q w.r.t. I and IC.

To a given query, we firstly identify whether the query is computable, and select a relevant algorithm according to the query type to compute CQA. To several tractable queries, we divide them into two types: key-key join query and nonkey-key join query, and present relevant algorithms.

Example 2. In a relation $R(\underline{x},y)$, assume a query $q= \exists x:R(\underline{x},10)$ and an instance $I1=\{(\underline{v1},10),(\underline{v1},5)\}$, repairs are $I2=\{(\underline{v1},10)\}$ and $I3=\{(\underline{v1},5)\}$, though $I1 \models q$, $I3 \not\models q$. In an instance $I4=\{(\underline{v1},10),(\underline{v1},5),(\underline{v2},10)\}$, repairs are $I5=\{(\underline{v1},10),(\underline{v2},10)\}$ and $I6=\{(\underline{v1},5),(\underline{v2},10)\}$, $I4,I5,I6 \models q$.

We may employ $\forall y:(R_1(\underline{x},y) \rightarrow y$ to indentify consistency for $y=10$, this should obtain the following sentence:

$$Q= \exists x:R(\underline{x},10) \wedge \forall y:R_1(\underline{x},y) \rightarrow y=10.$$

Obviously, assume a query $q= \exists x:R(\underline{x},y)$, its rewritten query is the following:

$Q= \exists x:R(\underline{x},y) \wedge \forall y^*:R_1(\underline{x},y^*) \rightarrow y=y^*$. where, y^* is a possible value.

Above queries are not joined, the following sentences should consider nonkey-key join queries.

Example 3. In a relation $R(\underline{x},y)$, assume a query $q= \exists x,y,w:R_1(\underline{x},y) \wedge R_2(\underline{y},w)$. In an instance $I1=\{R1(\underline{v1},a1), R1(\underline{v1},a2), R2(\underline{a1},b1)\}$, repairs are $I2=\{R1(\underline{v1},a1),R2(\underline{a1},b1)\}$ and $I3=\{R1(\underline{v1},a2), R2(\underline{a1},b1)\}$, though $I1 \models q, I3 \not\models q$. In an instance $I4=\{R1(\underline{v1},a1), R1(\underline{v1},a2), R2(\underline{a1},b1),R2(\underline{a2},b1)\}$, all nonkey values ($a1$ and $a2$) of $v1$ in $R1$ appear in $R2$. As a result, $I4 \models q$. This may employ a sentence $\forall y:(R_1(\underline{x},y) \rightarrow \exists w: R_2(\underline{y},w))$ to check , the rewritten query is a conjunction about the sentence and original sentence.

Q= $\exists$ x,y,w:$R_1(\underline{x},y) \wedge R_2(\underline{y},w) \wedge \forall$ y:($R_1(\underline{x}, y) \rightarrow \exists$ w:$R_2(\underline{y},w)$).

III. ALGORITHM

A. Identifying Rewritable Query

Now we present several algorithms for treating rewritable queries. The first algorithm identifies whether queries are tractable, the second one presents the query rewriting method. Since our method is based on the join graph, the third algorithm considers recursive computation for subnodes. Since the number of relations is limited and computation without data, the following algorithms are simple on computation complexity.

Algorithm 1. *judge-rewritable*(q,IC)
INPUT: a query $q(v)$: $\exists$ w:$R_1(\underline{x}_1,y_1) \wedge \ldots \wedge R_m(\underline{x}_m,y_m)$; a set of key constraintsIC
OUTPUT: whether the query is rewritable
BEGIN
  G=the join graph of q; R={R_1,…R_m};
  IF m=1 {RETRUN TURE; END PROCEDURE;}
  FOR n=1 TO m-1 DO
    FOR i=n+1 TO m DO
      IF $R[n]$=$R[i]$
        {IF the join type of $R[n]$ is nonkey-key with cyclic or nonkey-nonkey
          {RETRUN FALSE; END PROCEDURE;}
          ELSE {RETRUN TRUE; END PROCEDURE;}}
        IF $R[n] \neq R[i]$ AND $R[n]$does not exist a join with $R[i]$
          {skip current loop;}
        IF the join types of $R[n]$ and $R[i]$ are in {nonkey-key with cyclic , nonkey-nonkey, incomplete nonkey-key and full incomplete key-key} or nodes with in-degree 0 are greater than 1
          { RETRUN FALSE;END PROCEDURE;}
      ELSE { RETRUN TRUE;END PROCEDURE;}
    ENDFOR
  ENDFOR
END

Figure 2. An algorithm for identifying rewritable queries.

Since queries without joins are rewritable, algorithm 1 firstly identifies the number of relations in the input query q with built-in predicates v indicated as $\exists$ w:$R_1(\underline{x}_1,y_1) \wedge \ldots \wedge R_m(\underline{x}_m,y_m)$, x,y indicate a constant or a variable in the key and non-key position respectively, w, is a variable, v is a free variable.if it is a query without join, the algorithm returns TRUE and ends the procedure. The algorithm compares set elements of relations to identify whether self-join exist, if self-join exist, its join type is nonkey-key with cyclic or nonkey-nonkey, the algorithm returns FALSE and ends the procedure; otherwise, the algorithm returns TRUE and ends the procedure. If the query is not self-join., its join type is nonkey-key with cyclic, nonkey-nonkey, incomplete nonkey-key, full incomplete key-key or nodes with in-degree 0 are greater than 1, the algorithm returns FALSE and ends the procedure; otherwise, the algorithm returns TRUE and ends the procedure.

B. Rewriting Algorithm

If a query is writable, we should employ the following algorithm to obtain a rewritten query.

Algorithm 2 firstly obtains join components and their root nodes of the join graph w.r.t the query by cyclicing according to the number of nodes. If the root node of a join component exist subnodes, we should obtain join components of the subnodes of the node. If the node is a tree root, we should obtain the node value w.r.t the tree root and return $q \wedge \exists y_t$:$R_t(\underline{x}_t,y_t) \wedge \forall y_t$:$R_t(\underline{x}_t,y_t) \rightarrow recure(Q_t)$; $recure(Q_t)$ should call a recursive algorithm(Fig. 4). In "$q \wedge \exists y_t$:$R_t(\underline{x}_t,y_t) \wedge \forall y_t$:$R_t(\underline{x}_t,y_t)$", q is an original query and $R_t(\underline{x}_t,y_t)$ is a root node, conductional variables of existential quantifiers and universal quantifiers are nonkey attributes.

Algorithm 2. *rewritten_query*(q, IC)
INPUT: a query $q(v)$: $\exists$ w:$R_1(\underline{x}_1,y_1) \wedge \ldots \wedge R_m(\underline{x}_m,y_m)$;a set of key constraints IC
OUTPUT: a rewritten query
BEGIN
  G=the join graph of q
  FOR i:=1 to m DO
    $T_1,\ldots,T_m$=the join component of G;
    $R_i(\underline{x}_i,y_i)$=the root node of T_i;
    $Q_i(x_i,v)$=NULL;
    FOR j:=1 to m DO
      IF $R_j(\underline{x}_j,y_j)$ is a subnode of $R_i(\underline{x}_i,y_i)$
        {$Q_i = Q_i \wedge Q_j$;}
    ENDFOR
    IF $R_i(\underline{x}_i,y_i)$ is a tree root
      {$t=i$;}
  ENDFOR
  $Q=q \wedge \exists y_t$:$R_t(\underline{x}_t,y_t) \wedge \forall y_t$:$R_t(\underline{x}_t,y_t) \rightarrow recure(Q_t)$;
  RETURN Q
END

Figure 3. A query rewriting algorithm.

Algorithm 3 is a recursive algorithm, its input is query components, and its output is the rewritten query of the subnode. We firstly obtain the node and their subnodes that correspond to of query components. If the node is a leaf and nonkey attributes of the node, which exist the conductional variables of existential quantifiers in the original query, the algorithm returns $\exists$ x:$R(\underline{x},v)$ $\wedge$ $\forall$ v':($R(\underline{x},v')$ $\rightarrow$ v=v'). Conductional variables of existential quantifiers are the key attribute, conductional variables of universal quantifiers are nonkey attributes. If the node is a leaf and its nonkey attributes is equal to a constant, the algorithm returns $\exists$ x:$R(\underline{x},c)$ $\wedge$ $\forall$ v':($R(\underline{x},v')$ $\rightarrow$ v'=c), conductional variables of existential quantifiers are the key attribute, conductional variables of universal quantifiers are nonkey attributes. If the node is not a leaf, $\exists y$:$R(\underline{x},y) \wedge \forall y$:$R(\underline{x},y) \rightarrow recure(q)$, $recure(q)$ executes a recursive computation, conductional variables of existential quantifiers and universal quantifiers are nonkey attributes.

Algorithm 3. *recure*(q)
INPUT: a conjunctive query $Q^* \wedge$ ...
OUTPUT: a rewritten query of a subnode
BEGIN
  $R(\underline{x},y)$ is the node that corresponds to of q;
  $R(\underline{x},y)$ is the subnode of the node that corresponds to of Q^*;
  IF q=NULL and y=v /*if it is a leaf*/
    {$Q = \exists$ x:$R(\underline{x},v) \wedge$ $\forall$ v':($R(\underline{x},v') \rightarrow v$=$v'$);}

IF q=NULL 且 y_j =constant c /*if it is a leaf*/
{ $Q = \exists x{:}R(\underline{x},c) \wedge \forall v'{:}(R(\underline{x},v') \rightarrow v'=c)$;}
IF $q \neq$ NULL /*if it is not a leaf*/
{ $Q = \exists y{:}R(\underline{x},y) \wedge \forall y{:}R(\underline{x},y) \rightarrow recure(q)$;}
RETURN Q
END

Figure 4. A recursive query rewriting algorithm.

C. Correctness Proof

Theorem 10. To a given instance I and a set of key constaints Σ over a scheme R, a conjunctive query $q \in C_{PIME}$ over the scheme R, Q is a first-order query by algorithm 2. If $t \in I$, $I \models Q(t) \leftrightarrow t \in CQA(q,I,IC)$.

Proof: Assume G be a join graph of q, $T_1,\ldots,T_m$ are join components of G. Assume $q : \exists w{:}R(w,z)$, its detailed expression is the following: $\exists w_1,\ldots,w_m{:}R_1(w_1,z_1),\ldots,R_m(w_m,z_m)$, R is a conjunctive join relation set. To every $1 \leq i \leq m$, assume $R_i(\underline{x_i},y_i)$ be the root node of T_i, where, $w_i \notin x_i$ and $z_i \notin x_i$. Assume $q_i(x_i,z_i)= \exists w_i{:}R_i(x_i,w_i,z_i)$,$Q_i(x_i,z_i)= recure(q_i)$. Assume I^* is a repair of I.

($\Rightarrow$)If $I \models Q(t)$,exist a variable $v(z)=t$ and$I \models R(w,z)[v]$, to every $1 \leq i \leq m$, exist $v(x_i)=c_i$ and $v(z_i)=t_i(I \models Q(c_i, t_i))$. Assume $I^* \not\models q(t)$, $I^* \not\models q(v)$. Since there is not variable w_i that appear in $w_j(i \neq j,1 \leq i \leq m,1 \leq j \leq m)$, $I^* \not\models q_i(c_i,z_i)$, so $CQA(q_i(c_i,z_i),I,IC)$=FALSE, that is $I \not\models Q(c_i,z_i)$. Contradiction.

($\Leftarrow$)If $t \in CQA(q,I,IC)$. Assume $I \not\models Q(t)$ and exist a variable $v(z)=t$.If$I \not\models q(z)[v]$, since $I^* \in I$, $I^* \not\models q(z)[v]$; if $I \not\models Q_i(x_i,z_i)[v](1 \leq i \leq m)$, $CQA(q_i(x_i,z_i)[v],I,IC)$=FALSE, so $I^* \not\models q_i(x_i,z_i)[v]$, that is $I^* \not\models q(z)[v]$. As a result, to every variable $v(z)=t$, exist $I^* \not\models q(z)[v]$, thus $t \notin CQA(q,I,IC)$. Contradiction.

IV. Conclusion

This paper proposes an inconsistent data management theory based on the first-order logic and the directed join graph for identifying intractable and tractable queries by analyzing join completeness of polytype queries. To a great number of practical tractable conjunctive queries, we present several query rewriting algorithms to construct a new rewritten query by recursive computation according to the join graph w.r.t. an original query. The next work should consider more intergrity types such as foreign key type.

Acknowledgements

This work is supported by Scientific Research Fund of Hunan Provincial Education Department of China under Grant No. 08B040; Scientific Research Fund of Loudi Science& Technology Department (2012).

Rrferences

[1] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," Proceedings of ACM Symposium on Principles of Database Systems.New York: ACM Press,1999,pp. 68-79.
[2] D. Xie and L. M. Yang, "Study on consistent query answering in inconsistent databases," Frontiers of Computer Science in China,vol.1,pp.493-501, 2007.
[3] T. Dasu and T. Johnson, "Exploratory Data Mining and Data Cleaning," New York: John Wiley, 2003.
[4] Fuxman A and Miller R J, "First-order query rewriting for inconsistent databases.Journal of Computer and System Sciences,"vol.73,pp.610−635, 2007.
[5] Wijsen J, "Consistent query answering under primary keys: a characterization of tractable queries," Proceedings of ACM Symposium on Principles of Database Systems.New York: ACM Press, 2009,pp.42-52.
[6] Wijsen J, "On the First-order expressibility of computing certain answers to conjunctive queries over uncertain databases," Proceedings of ACM Symposium on Principles of Database Systems.New York: ACM Press,2010,pp.179-190.
[7] Transaction Processing Performance Council, "TPC Benchmark H standard specification," http://www.tpc.org,2011.
[8] J Chomicki,Marcinkowski J, and Staworko S, "Computing consistent query answers using conflict hypergraphs," Proceedings of the International Conference on Information and Knowledge Management.New York: ACM Press, 2004, pp.417-426.
[9] Caniupan M and Bertossi L,"The consistency extractor system: answer set programs for consistent query answering in databases," Data & Knowledge Engineering, vol.69,pp.545-572,2010.
[10] Chomicki J and Marcinkowski J, "Minimal-change integrity maintenance using tuple deletions," Information and Computation, vol.197,pp.90-121,2005.

Dong XIE is an associate precefessor of Hunan University of Humanities, Science and Technology, Loudi, P.R. China. He earned his Ph.D. degree in computer appcation technology from Central South University, Changsha, China, in 2007. His research interests include data management, internet of things and supply chain.
Xinbo Chen is a lecture of Loudi Vocational and Technical College, Loudi, P.R. China. He earned his master degree in computer appcation technology. His research interests include data management and network technology.
Yan Zhu is a lecture of Loudi Vocational and Technical College, Loudi, P.R. China. She earned her master degree in computer appcation technology. His research interests include data management and network technology.