IEEE-COMPSAC 1991, Kyoto

Implicit Joins in the Structural Data Model

Byung Suk Lee
Bellcore

Witold Litwin
INRIA.

Gio Wiederhold
Stanford University

Abstract

In general, it is cumbersome and prone to errors to write a relational query involving many join predicates. One approach to resolve this problem is the implicit join method [6, 8]. This method derives the unspecified join predicates of an incompletely specified query from the semantic dependency - a set of the pairs of semantically dependent attributes - in the database schema. Although the implicit join method has some advantages over the universal relation approach [2, 4, 5], it frequently derives a complete query differently from what a user would do, and generates redundant subqueries. This is because, in the implicit join method, the semantic dependency is discovered by a syntactic observation of the database schema. We show that using the structural model [10, 11, 12] removes the problem of redundancy by making the semantic observation of the database schema possible, and provides a complete query that many users would prefer to that of the original implicit join method.

1 Introduction

The relational model has been widely accepted since it was proposed by Codd [1]. The relational algebra is equivalent to the relational calculus in terms of relational completeness [2]. A join is expressed as a join predicate in the relational calculus. Joins are essential in a relational query. They transform the set of relations referred to by the query to a single relation the query defines. To specify the joins basically requires performing logical navigation over relations. It is however often cumbersome to specify all the joins participating in the query. This has motivated a lot of research on user interfaces for formulating a relational query without joins. A graphical interface and natural language interface fall into this category. Another category are techniques for writing a relational query without explicitly specifying some joins. One approach is the universal relation [2, 4, 5] scheme. It eliminates joins from the user query by presenting to users a single relation (universal) schema from all relations in the database. Another approach in [6, 8] derives the unspecified joins, called implicit joins, from a database schema or from the multiple schemas for queries with interdatabase joins. The method has advantages over the universal relation approach, easy updating in particular. There are also cases when the query the method provides appears preferable to the query that would result from the universal relation approach [6].

In [6] implicit join scheme uses semantic dependency - a set of the pairs of semantically dependent attributes - to derive implicit joins from the database schema and from explicit joins that are the joins specified in the query. A query completion algorithm generates minimal complete queries given an incomplete query. Semantic dependency configures into the edges of a database graph. Given an incomplete query and its query graph, the database graph is searched to find minimal trees covering all the nodes in the query and including all explicit joins. Each of the minimal trees is translated into a complete subquery. The final query is the union of the complete subqueries.

Semantic dependency is sufficient for deriving implicit joins. However, sometimes it contains more than necessary dependency information so that the query completion algorithm generates needlessly equivalent, alternative complete queries. This is because in [6] a semantic dependency is discovered by syntactic observation of the database schema . We discuss in Section 2.3 a situation when one gets ten queries, while four are enough.

We propose in what follows a solution to this problem, that is the use of the structural data model [10, 11, 12]. In this model, the semantic dependency is defined through explicitly declared connections. We will call this dependency *explicit semantic dependency* (ESD) while the *implicit semantic dependency* (ISD) will be this from [6]. We prove that the use of the ESD allows us:

- to avoid redundant subqueries.
- to have, for the corresponding non-redundant subqueries, no more joins than the complete queries of the original method, provided the database satisfies some rules of referential integrity we define.
- to provide the complete queries that many users would prefer to those of the original method.

The use of the structural model thus makes the processing of queries with implicit joins more efficient. Our work constitutes also a method simplifying queries to the structural data model with respect to the previous use of this model.

In Section 2, we review the concept of semantic dependency and the computation of implicit joins as described in [6]. In Section 3, we review the structural model. Then, in Section 4, we discuss using the semantics of connection constraints for deriving implicit joins. Finally, Section 5 contains the conclusion.

2 Implicit joins

2.1 Semantic dependency

The idea of implicit joins is to avoid specifying all join predicates in the query, and have the system derive the missing join predicates. The system needs information complementary to the missing join predicates. This information is called the semantic dependency in [6]. It is classified into *natural dependency* and *equivalence dependency*. The natural dependency is discovered from the usual database schema, and is used to derive implicit joins within a single database. The equivalence dependency is assumed to be defined in databases in multidatabase environment in addition to the usual definition of relations, and is used to derive implicit joins between databases. We describe these two dependencies here.

In the following discussion, DOM(X) denotes the cartesian product of the domains of each attribute of X, Key(R) denotes the primary key of a relation R, and Key$'$(R) denotes a candidate key [3] of a relation R.

Naturally dependent attributes are found according to the following definition.

Definition 2.1 (Natural dependency) Given two sets of attributes $A \subseteq R_1$ and $B \subseteq R_2$ where R_1 and R_2 are relations on the *same* database, A and B are said to be naturally dependent if and only if DOM(A) = DOM(B) and $A \subseteq$ Key(R_1) $\vee B \subseteq$ Key(R_2), that is, A and B share a domain and at least one belongs to the primary key. A natural dependency is a set of the pairs of naturally dependent attributes.

This dependency allows us to consider that the same value of an attribute in different relations correspond to the same thing [6].

The notion of domain equivalence needed for natural dependency is not valid precisely for defining an equivalence dependency because the equivalence dependency is used in multidatabase environment. For example, there frequently occurs a domain mismatch problem [9] between attributes in different databases. Hence, we introduce a notion of *approximate* domain equivalence, denoted as DOM(A) $\approx$ DOM(B), which acknowledges the possibility of domain mismatch and assumes a necessary mismatch resolution mechanism. Sometimes the domain mismatch also hinders the mapping of primary keys between two relations on different databases. In [6], secondary keys are used in such a case. Likewise, we use a *candidate* key for the definition of equivalence dependency here.

Definition 2.2 (Equivalence dependency) Given two sets of attributes $A \subseteq R_1$ and $B \subseteq R_2$ where R_1 and R_2 are relations on *different* databases, A and B are said to be equivalently dependent if and only if DOM(A) $\approx$ DOM(B) and $A \subseteq$ Key$'$(R_1) $\wedge B \subseteq$ Key$'$(R_2). An equivalence dependency is a set of the pairs of equivalently dependent attributes.

This dependency relates the sets of domains in different databases where identity of values implies the same real object [6].

Inventory Multidatabase Schema.
Underlined attributes are the primary key attributes.
IA (Inventory database A):
S(<u>s#</u>, name, tel, city) /* Supplier */
P(<u>p#</u>, name, color) /* Part */
SP(<u>s#</u>, <u>p#</u>, qty) /* Supplier-Part */
Comp(<u>majp</u>, <u>minp</u>, assembly-type) /* Component */

IB (Inventory database B):
Sup(<u>id</u>, name, phone#, addr) /* Supplier */
Itm(<u>inum</u>, name, size, weight) /* Item */
SupItm(<u>id</u>, <u>inum</u>, unit-price, quantity)

Figure 1: Multidatabase schema

Example 2.1 The relations of the IA (inventory database A) whose schema is shown in Figure 1 have the following natural dependency: { (S.s#, SP.s#), (SP.p#, P.p#), (SP.p#, Comp.majp), (SP.p#, Comp.minp), (P.p#, Comp.majp), (P.p#, Comp.minp) }. Similarly, IB has the following natural dependency: { (Sup.id, SupItm.id), (SupItm.inum, Itm.inum) }. We further consider there is the following equivalence dependency between the relations of the two different databases IA and IB: { (IA.S.name, IB.Sup.name),(IA.P.name, IB.Itm.name) }, assuming that name is a secondary key of IA.S, IA.P, IB.S, IB.Itm, and domains are approximately equivalent between IA.S.name and IB.Sup.name, and between IA.P.name and IB.Itm.name. □

The natural dependency and equivalence dependency are not exhaustive. However, we ignore the other (trivial) dependency as semantically meaningless [6]. We say a pair of attributes are semantically dependent if and only if they are either naturally dependent or equivalently dependent.

2.2 Computation of implicit joins

Given the dependency information, the system generates missing join predicates using a *query completion algorithm* shown in Algorithm 2.1. For simplicity, we restrict the discussion here to conjunctive select-project-equijoin queries. A query is either a monodatabase or multidatabase query. We use the MSQL [8] as our query language. See [6] for more complete discussion of the query completion algorithm.

We first define two graphs – multidatabase graph and query graph – and then describe the query completion algorithm.

Definition 2.3 (Multidatabase graph) A multidatabase graph is an undirected connected graph where each node represents a relation and each edge represents an element of the semantic dependency. An edge represents an element of the natural dependency if it is connecting two nodes of the same database, and equivalence dependency if different databases.

An example of the multidatabase graph is shown in Figure 2 for the Inventory multidatabase in Figure 1. Attribute names are shown for clarity.

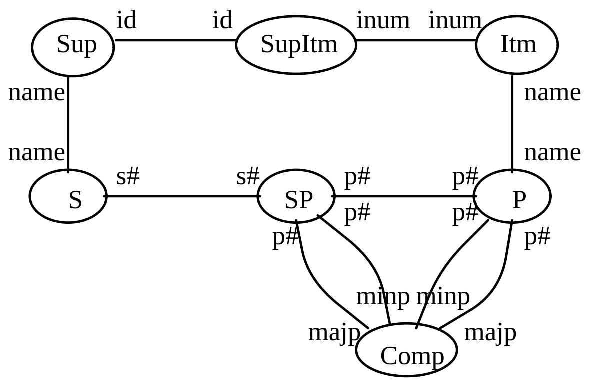


Figure 2: Multidatabase graph

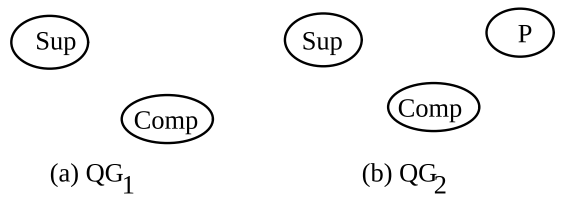


Figure 3: Incomplete query graphs (QG_1 and QG_2)

Definition 2.4 (Query graph) A query graph is an undirected graph where each node represents a relation and an edge represents an explicit join appearing in the query. If there is a join between two instances of the same relation, they are distinguished by a tuple variable-like subscript.

Example 2.2 Given the multidatabase of Figure 1, let us assume users issue the following incomplete multidatabase queries.

Q_1: Select addr where assembly-type = 'screw';

Q_2: Select addr, color where assembly-type = 'screw';

The query graphs of Q_1 and Q_2 are shown in Figure 3. Note these queries do not have *From* clauses [8]. For example, the user's thought for Q_1 is "I don't know the structure of the databases. All I know is that the assembly type is screw. Now, I want to know the suppliers' addresses *corresponding to* the assembly type." □

In general, the graph of an incomplete query is a disconnected graph. We call the edges and nodes that appear in an incomplete query graph as *explicit edges* and *explicit nodes*. It is the goal of the query completion algorithm to find out the *impilcit edges* and *implicit nodes* that render the query graph complete.

Algorithm 2.1 (Query completion)
Input:multidatabase graph(DG),incomplete query(Q)
Output: complete query (Q')

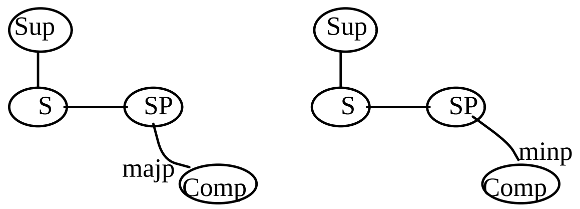


Figure 4: Complete query graphs (QG'_1)

```
Procedure:
  Let $E_e$ be the set of explicit join edges in QG.
 1. Parse the query Q into a query graph QG.
 2. For each node $v$ of DG which has multiple instances (distinguished by subscripts) in the QG,
   a. Replicate $v$ and the edges incident to $v$ so that DG has the same number of the instances of $v$ as QG.
   b. Distinguish the replicated nodes with a subscript, i.e., $v_1, v_2, \cdots, v_n$ where $n$ is the number of the instances of $v$ in QG.
 3. Find all connected subgraphs of DG such that each subgraph includes all the nodes of QG and has minimal number of edges.
 4. For each subgraph $(V', E')$ found in Step 3,
   a. Compute the set of implicit join edges as $E_i := E' - E_e$.
   b. Add to the condition part of Q the conjunction of all join predicates corresponding to the elements of $E_i$, and set the base relation part of Q to the relations corresponding to the elements of $V'$.
```

The subgraphs found in Step 3 are all minimal trees and have the same number of edges and nodes. Users are assumed to want the union of the complete queries formulated in Step 4.

2.3 Examples of query completion

We use the Inventory multidatabase (Figure 1) whose database graph was shown in Figure 2.

Let's consider the incomplete query Q_1 of Example 2.2. Its complete query graphs (QG'_1) obtained by Algorithm 2.1 are shown in Figure 4. The resulting complete queries are:

- Select IB.Sup.addr from IB.Sup, IA.S, IA.SP, IA.Comp where IA.Comp.assembly-type = 'screw' and IA.Comp.majp = IA.SP.p# and IA.SP.s# = IA.S.s# and IA.S.name = IB.Sup.name;
- Select IB.Sup.addr from IB.Sup, IA.S, IA.SP, IA.Comp where IA.Comp.assembly-type = 'screw' and IA.Comp.minp = IA.SP.p# and IA.SP.s# = IA.S.s# and IA.S.name = IB.Sup.name;

Let's consider the incomplete query Q_2 of Example 2.2. Algorithm 2.1 generates ten complete queries whose graphs are shown in Figure 5. Among these ten queries, (a), (b), (c) are equivalent and (e), (f), (g) are equivalent because of equijoin associativity. Therefore, there is a problem of partitioning the set of complete

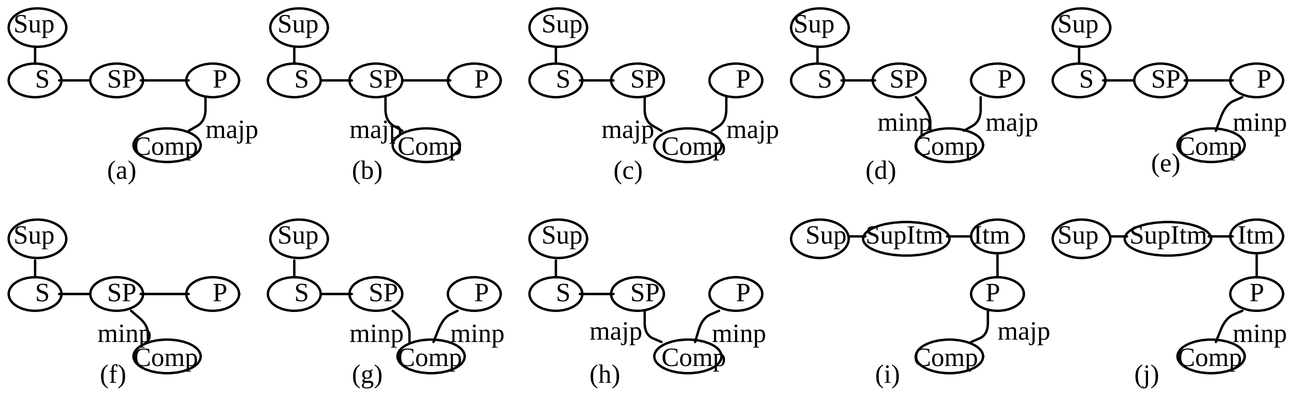


Figure 5: Complete query graphs (QG_2')

queries into a set of equivalence classes and choosing one from each class.

3 Structural model

The structural model is the relational model augmented with connections [10, 11, 12]. Connections are part of the data model and used to enforce interrelational constraints - cardinality constraints and update constraints. The structural model consists of seven relation types (entity relations, foreign entity relations, nest relations, associative relations, lexicons, subrelations, derived relations) and four connection types (ownership connection, reference connection, subset connection, identity connection). Relations are those of the relational model and take on different types depending on the connections to other relations.

3.1 Connections

Table 1 shows the constraints of the four connection types [13]. Each connection type specifies its own update constraint. These constraints preserve the referential integrity between connected tuples. For example, for an ownership connection $R_1.A_1 \longrightarrow\!\!* \; R_2.A_2$, there must exist a matching tuple in R_2 for each tuple of R_1 such that $R_1.A_1 = R_2.A_2$. We denote the referential integrity constraint as $R_1.A_1 \rightarrow R_2.A_2$ for all types of connections. In Table 1, *Cardinality* column shows the cardinality relationship between the source relation and the destination relation, and the arrows in the *Ref. int.* column show the directions in which the referential integrity constraints are enforced.

By convention, ownership, reference, and subset connections consider a single database only. The identity connection is distinct from the others in that it is used to enforce (delayed) update consistency of derived data or of replicated data in distributed databases [13].

Formally describing, given two relations R_1 and R_2, a connection from $A \subseteq R_1$ to $B \subseteq R_2$ satisfies the constraints of Definition 3.1. Connections are defined between two sets of attributes, at least one of which should be a primary key. One can interpret it as a formal description of an entity-relationship model [14] without many-to-many relationships. Connections also make it possible to describe entities in a more object-oriented manner than when we have relations only [15]. In this way, connections support the semantics of 'correspondence' between database objects as used in [6].

Definition 3.1 (Connection constraints)

- Ownership connection: $\mathrm{DOM}(A) = \mathrm{DOM}(B) \wedge A = \mathrm{Key}(R_1) \wedge B \subseteq \mathrm{Key}(R_2) \wedge A = B$
- Reference connection: $\mathrm{DOM}(A) = \mathrm{DOM}(B) \wedge B = \mathrm{Key}(R_2) \wedge (A = \mathrm{null} \vee A = B)$
- Subset connection: $\mathrm{DOM}(A) = \mathrm{DOM}(B) \wedge A = \mathrm{Key}(R_1) \wedge B = \mathrm{Key}(R_2) \wedge A = B$
- Identity connection: $\mathrm{DOM}(A) = \mathrm{DOM}(B) \wedge A = \mathrm{Key}(R_1) \wedge B = \mathrm{Key}(R_2) \wedge A =_d B$ where d is the time delay.

3.2 Using the identity connection in a multidatabase

As mentioned in Section 3.1, the identity connection is defined for a (delayed) synchronous update of replicated data in a single or distributed database. This concept needs to be modified to fit into multidatabase environment. In a multidatabase, an identity connection is defined between attributes belonging to different databases. Hence, we should use the same notion of approximate domain equivalence and a candidate key as in Definition 2.2. Synchronous updates between connected attributes are not needed if we assume each database operates autonomously. Hence, we drop the constraint of $A =_d B$ from the constraints of Definition 3.1. Thus, the constraint of the identity connection used in multidatabase environment is as follows.

Definition 3.2 (Identity connection constraint) $\mathrm{DOM}(A) \approx \mathrm{DOM}(B) \wedge A = \mathrm{Key}'(R_1) \wedge B = \mathrm{Key}'(R_2)$

Type	*Cardinality*	*Symbol*	*Insertion Rule* *Deletion Rule*	*Ref. int.*
Ownership	1:m	—∗	Source (Owner) tuple must exist. Source deletion implies Destination deletion.	←
Reference	n:1	>—	Destination (Referenced) tuple must exist. Source existence prohibits Destination deletion.	→
Subset	$1{:}1_{partial}$	—∋	Source (Super) tuple must exist. Source deletion implies Destination deletion.	←
Identity	$1{:}1_{delayed}$	=—=	Source (Primary) tuple must be updated first. Source deletion causes Destination deletion.	↔

Table 1: Connections of the structural model

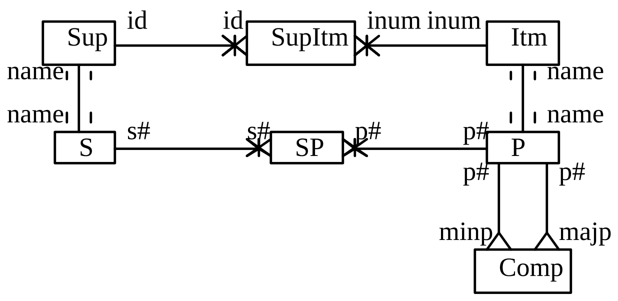


Figure 6: Structural diagram

Example 3.1 Figure 6 shows the diagram of the structural model of the Inventory multidatabase (Figure 1). The following types of connections are shown.

- Ownership: S.s# —∗ SP.s#, P.p# —∗ SP.p#, Sup.id —∗ SupItm.id, Itm.inum —∗ SupItm.inum
- Reference: Comp.majp >— P.p#, Comp.minp >— P.p#
- Identity: S.name =—= Sup.name, P.name =—= Itm.name

There is no connection between Comp.majp and SP.p# and between Comp.minp and SP.p# since there is no referential integrity constraint between them. □

4 Implicit joins in the structural model

4.1 Connection constraints as semantic dependency

A case was shown in Section 2.3 in which ISD generates redundantly equivalent queries. Now we show this problem can be resolved by using the ESD. We start from the following Lemma.

Lemma 4.1 Given two sets of attributes $A \subseteq R_1$ and $B \subseteq R_2$ where R_1 and R_2 are relations,

- If R_1 and R_2 are on the same database, and A and B are connected by an ownership, reference, or a subset connection, then A and B are naturally dependent.
- If R_1 and R_2 are on different databases, and A and B are connected by an identity connection, then A and B are equivalently dependent.

Proof: Let us compare Definition 2.1 and Definition 2.2, respectively, with Definition 3.1 and Definition 3.2. The constraints of an ownership, reference, and subset connections (Definition 3.1) all satisfy the condition of natural dependency (Definition 2.1) between two relations on the same database. Likewise, the constraints of an identity connection in multidatabase environment (Definition 3.2) satisfies the condition of equivalence dependency (Definition 2.2). Note the contrary is *not* always true. Q.E.D.

The semantic dependency defined by Lemma 4.1 is an ESD while that of Section 2.1 is an ISD.

Lemma 4.2 Given a database schema, the ESD is a subset of the ISD, i.e., ESD ⊆ ISD.

Proof: According to Lemma 4.1, any pair of attributes that have connections between them are semantically dependent. However, all semantically dependent pairs do not necessarily have connections between them unless they have referential integrity to be preserved. Q.E.D.

We can now deduce the following theorem.

Theorem 4.1 The number of equivalent complete queries generated using ESD is no more than the number of those generated using ISD.

Proof: The number of complete queries is decided in Step 3 of Algorithm 2.1. Let's say we have a set G_I of complete query graphs generated using ISD. By Lemma 4.2, we know there exist $G'_I \subseteq G_I$ where G'_I is a set of zero or more $g \in G_I$ that contain edges corresponding to the elements of ISD that do not belong to ESD. Those g's cannot appear as a connected graph in the set G_E of complete query graphs generated using ESD. Therefore, $G_E = G_I - G'_I$. Since G'_I has zero or more elements, G_E has no more elements than G_I. Q.E.D.

Thus, if we complete queries using ESD only, we can avoid the overhead of generating equivalent complete queries and choosing one of them.

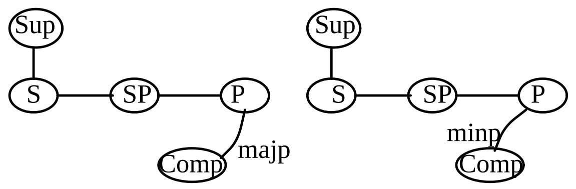


Figure 7: Complete Query graphs (QG'_3)

Example 4.1 Let's consider the Inventory multidatabase whose structural diagram was shown in Figure 6. The structural diagram can be viewed as a multidatabase graph as well.

Given the incomplete query Q_1 of Example 2.2, the complete query graph (QG'_3) obtained with ESD is shown in Figure 7. QG'_3 has more number of joins than QG'_1. In Example 4.2, it will be reduced to the same number.

Given the incomplete query Q_2 of Example 2.2, the complete query graphs (QG'_4) obtained with ESD is the same as the a, e, i, and j of QG'_2 shown in Figure 5. Note the other six equivalent alternative complete queries are not generated because of the missing connections between SP and Comp. Generating less alternatives will reduce the overhead of choosing one of them. □

4.2 Reducing a complete query to a simpler query

We now show that it is possible to reduce a complete query to a simpler query using the semantics of referential integrity defined by connections.

We say a join path follows a *chain* of connections if, for each relation on the path, the connections to the relation are defined on the *same attribute*. Besides, the referential integrity constraint along a chain of connections, $(R_1.A_1 \rightarrow R_2.A_2) \wedge (R_2.A_2 \rightarrow R_3.A_3) \wedge \cdots (R_{n-1}.A_{n-1} \rightarrow R_n.A_n)$, is denoted by a simpler form $R_1.A_1 \rightarrow R_2.A_2 \rightarrow \cdots \rightarrow R_n.A_n$ and called *a chain of referential integrity.*

A query can be *reduced* if we can replace a join path with a single join and still get the same tuples. The following lemma describes the conditions that make such a reduction possible.

Lemma 4.3 Let's say there is a join path following connections $C_{i,i+1}, i = 1, 2, \cdots, n-1$, where $C_{i,i+1}$ is a connection between $R_i.B_i$ and $R_{i+1}.A_{i+1}$. Then

$$\Pi_{R_1 \cup R_n}(R_1 \underset{B_1 = A_2}{\bowtie} R_2 \underset{B_2 = A_3}{\bowtie} \cdots \underset{B_{n-1} = A_n}{\bowtie} R_n) = R_1 \underset{B_1 = A_n}{\bowtie} R_n \qquad (1)$$

if and only if

$$A_i \equiv B_i, i = 2, 3, \cdots, n-1 \qquad (2)$$

i.e., the join path follows a chain of connections, and

$$\exists R_p, 1 \leq p \leq n, \\ (R_1.A_1 \rightarrow \cdots \rightarrow R_p.A_p \wedge \\ R_p.A_p \leftarrow \cdots \leftarrow R_n.A_n) \qquad (3)$$

i.e., the chain of referential integrity is either unidirectional or can be decomposed into two subchains heading toward each other.

Proof: Let's define T_E as the LHS of Equation 1 and T_I as the RHS.

1. *if* part: Given Equation 2, T_E becomes $\Pi_{R_1 \cup R_n}(R_1 \underset{A_1 = A_2}{\bowtie} R_2 \underset{A_2 = A_3}{\bowtie} \cdots \underset{A_{n-1} = A_n}{\bowtie} R_n)$. Then we know, by the definition of joins, $t \in T_E$ if and only if

$$\exists t_2 \in R_2, t_3 \in R_3, \cdots, t_{n-1} \in R_{n-1}, \\ (t.A_1 = t_2.A_2 \wedge t_2.A_2 = t_3.A_3 \wedge \cdots \wedge \\ t_{n-1}.A_{n-1} = t.A_n) \qquad (4)$$

for the given t. Similarly we know $t \in T_I$ if and only if

$$t.A_1 = t.A_n \qquad (5)$$

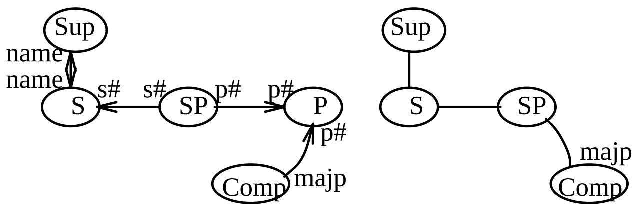
for the given t. Then, it can be easily seen that Equation 4 becomse equivalent to Equation 5 given that Equation 3 is true.

2. *only if* part: Let's prove this by contradiction. First, let's assume Equation 2 is not satisfied although Equation 1 is true. We know, by the definition of joins, $t \in T_E$ if and only if $\exists t_2, t_3, \cdots, t_{n-1}(t.B_1 = t_2.A_2 \wedge t_2.B_2 = t_3.A_3 \wedge \cdots \wedge t_{n-1}.B_{n-1} = t.A_n)$. Likewise, $t \in T_I$ if and only if $t.B_1 = t.A_n$. From these, it is clear that $T_E \neq T_I$. This contradicts the assumption.

Second, let's assume Equation 3 is not satisfied, although Equation 1 and Equation 2 are true. The following shows the examples of such a case for a join path between R_1 and R_n.

1. $R_1 \longleftarrow R_i \longrightarrow R_n$
2. $R_1 \longleftarrow R_i \longrightarrow R_j \longleftarrow R_n$
   $R_1 \longrightarrow R_i \longleftarrow R_j \longrightarrow R_n$
3. $R_1 \longleftarrow R_i \longrightarrow R_j \longleftarrow R_k \longrightarrow R_n$
   $R_1 \longrightarrow R_i \longleftarrow R_j \longrightarrow R_k \longleftarrow R_n$
4. . . .

(The R_i, R_j, and R_k are intermediate nodes on the join path and switching the direction of the chain of referential integrity. The numbers on the left side are the number of these intermediate nodes. The arrows denote the chains of referential integrity along connections.) The examples show that any case not satisfying Equation 3 has at least one intermediate node from which the chains of referential integrity are *going out.* Therefore, in such a case, there exist three nodes $R_{p_1}, R_{p_2}, R_{p_3}$ for $1 \leq p_1 < p_2 < p_3 \leq n$ such that neither $R_{p_1}.A_{p_1} \rightarrow R_{p_2}.A_{p_2}$ nor $R_{p_2}.A_{p_2} \leftarrow R_{p_3}.A_{p_3}$ is true. From this observation, we can see that Equation 4 is not equivalent to Equation 5. In other words, $t \in T_E$ is not equivalent to $t \in T_I$. This contradicts the assumption. Q.E.D.



(a) Before reduction (b) After reduction

(Arrows shown in (a) denote the referential integrity. Sup and Comp are the only explicit nodes.)

Figure 8: Reducing QG_3' to QG_1'

Thus, as long as there is a chain of referential integrity for a join path, and it satisfies Equation 3, a query can be safely reduced to another one involving less number of joins.

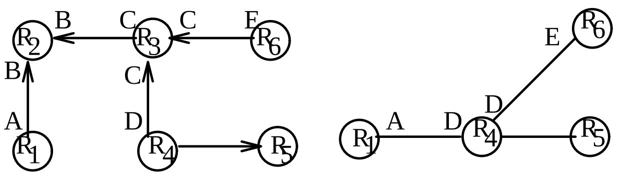
Based on Lemma 4.3, Algorithm 4.1 shows how to reduce a complete query. Note in Lemma 4.3, the LHS of Equation 1 was projected on $R_1 \cup R_n$. This was to make it compatible with the RHS. In the context of query completion, they are compatible only if $R_2, R_3, \cdots, R_{n-1}$ are implicit nodes.

Algorithm 4.1
Input: a complete query (QG')
Output: a reduced complete query (QG'')
Procedure:

1. Find all pairs of nodes (R_i and R_j) that are connected by a join path P_{ij} of length > 1 which follows a maximal length chain of connections that satisfies Equation 3, and whose nodes between R_i and R_j are implicit nodes.
2. For each pair of nodes (R_i, R_j) found in Step 1, replace P_{ij} with a single edge between R_i and R_j.

Example 4.2 QG_3' shown in Figure 7 is reduced to QG_1' shown in Figure 4 by Algorithm 4.1. Figure 8 shows it for one of the complete queries. In Step 1, we find only one pair of nodes (SP, Comp), which is connected by the join path SP $\underset{\mathrm{p\#=p\#}}{\bowtie}$ P $\underset{\mathrm{p\#=majp}}{\bowtie}$ Comp. This is replaced by a single join SP $\underset{\mathrm{p\#=majp}}{\bowtie}$ Comp in Step 2. Note there is no referential integrity constraint between Sup and S because we assumed no (delayed) synchronous update in multidatabase environment. QG_3' and QG_1' produce the same tuples, i.e., this reduction is safe. This is because of the referential integrity constraints SP.p# $\rightarrow$ P.p# and Comp.majp $\rightarrow$ P.p#.

Figure 9 shows another example. We can find a pair of nodes (R_1, R_4) connected by the join path $R_1 \underset{A=B}{\bowtie} R_2 \underset{B=C}{\bowtie} R_3 \underset{C=D}{\bowtie} R_4$, which is replaced by $R_1 \underset{A=D}{\bowtie} R_4$. Likewise, the join path connecting R_4 and R_6 can be replaced by $R_4 \underset{D=E}{\bowtie} R_6$. Note b is not the only possible result. For example, the join path



(R_1, R_5, and R_6 are explicit nodes; R_2, R_3, and R_4 are implicit nodes. All edges are implicit. The arrows denote the direction of referential integrity constraint.)

Figure 9: Query reduction

between R_1 and R_6 could have been replaced by the join $R_1 \underset{A=E}{\bowtie} R_6$. $\Box$

Now we can deduce the following theorem.

Theorem 4.2 Complete queries generated using ESD and reduced by Algorithm 4.1 has no more joins than the complete queries generated using ISD provided that, for any $(R_i.A_i, R_j.A_j) \in$ ISD $-$ ESD, the join path between R_i and R_j generated using ESD satisfies Equation 2 and Equation 3 of Lemma 4.3.

Proof: Let's consider a complete query graph generated using ESD. From Lemma 4.2, $\Delta \equiv$ ISD $-$ ESD has zero or more elements. Therefore, adding the edges corresponding to the elements of Δ to ESD, we may find a 'short-cut' to a join path. Let's say we found a short-cut to a join path P_{ij} between R_i and R_j. Given that P_{ij} satisfies Equation 2 and Equation 3, we can consider a join path $P_{ik} \supseteq P_{ij}$ which satisfies the same conditions. Then P_{ik} will be replaced with a sigle edge by Algorithm 4.1. On the other hand, if ISD was used, $P_{ik} = \{(R_i, R_j)\} \cup P_{jk}$, i.e., there would be one or more joins between R_i and R_k. This completes the proof. Q.E.D.

4.3 Semantics of a complete query

Now, we discuss the case of the use of ESD to generate a query whose semantics is different from that generated using ISD.

Consider the structural diagram in Figure 10 and its relation schema. Let us consider now the incomplete query:

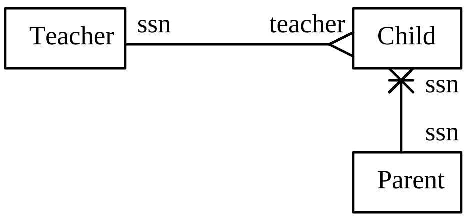
- Select t_name where p_name = 'Litwin'.

Using ISD we generate the complete query:

- Select Teacher.t_name from Teacher, Parent where Teacher.ssn = Parent.ssn and Parent.p_name = 'Litwin'.

This query will provide all the teachers who are the same person as the parent named Litwin, i.e., Litwin itself. On the other hand, using ESD we generate the query:

- Select Teacher.t_name from Teacher, Child, Parent where Teacher.ssn = Child.teacher and Child.ssn = Parent.ssn and Parent.p_name = 'Litwin'.



(a) Structural diagram

```
Teacher(ssn, t_name)
Child(ssn, c_name, teacher)
Parent(ssn, p_name)
```

(b) Relation schema

Figure 10: Example database schema

The corresponding semantics is 'Get the teachers who are teaching the children of the parent named "Litwin" '. This semantics is likely to be preferred by users.

5 Conclusion

The connections of a structural data model allows it to derive implicit joins more efficiently. The query completion algorithm can avoid generating redundant subqueries through the use of the connections as the source of semantic dependency. The remaining complete queries can be reduced to equivalent simpler queries that have no more joins than the complete queries of the original method. Our method can also provide complete queries that users prefer to those of the original method.

Our approach through the structural data model is a new alternative to both the implicit join approach and the universal relation approach. Its drawback is that it requires additional definitions in the database schema, i.e., those of the connections. It pays off however not only through a more efficient implicit join interface, but also other benefits such as explicit update constraints and integrity control rules [12].

With respect to further research, we need more work on the query reduction algorithm. Sometimes it produces alternative reduced queries, so it has room for further optimization. The algorithm considers in particular the number of joins as the only criterion. We are able to get a more optimal query by considering other criteria. For instance, we can consider also the relation cardinality and choose the query with the smallest relations.

References

[1] Codd, E, "A relational model of data for large shared data banks," in 'Readings in Database Systems' edited by Michael Stonebraker, pp. 5 - 15, Morgan Kaufmann Publishers, Inc., 1988.

[2] Ullman, J., "Principles of database systems" (2nd edition), Computer Science Press, Inc., 1983.

[3] Ullman, J., "Principles of Database and Knowledge-Base Systems", Vol.1, Computer Science Press, 1988.

[4] Maier, D., Ullman, J., and Vardi, M., "On the foundations of the universal relation model," ACM Transactions on Database Systems, Vol. 9:2, 1984.

[5] Lecluse, C., and Spyratos, N., "Implementing queries and updates on universal scheme interfaces," Proceedings of the 14th VLDB Conference, Los Angeles, California, 1988.

[6] Litwin, W., "Implicit joins in the multidatabase system MRDSM," Proceedings of the IEEE Computer Software & Applications Conference (COMPSAC), Chicago, Illinois, pp. 495 - 504, October 1985.

[7] Codd, E., "Relational database: a practical foundation for productivity," The Relational Journal, November, 1981.

[8] Litwin, W., "MSQL: a multidatabase language," Information Science (Special Issue on Databases), Vol. 48, No. 2, July 1989.

[9] DeMichiel, L., "Performing Operations over Mismatched Domains," Proceedings of the IEEE Data Engineering Conference 5, Los Angeles, February, 1989.

[10] El-Masri, R., "On the design, use, and integration of data models," Ph.D. dissertation, Computer Science Department, Stanford University, Stanford, CA 94305, May 1980.

[11] Wiederhold, G., "Design concepts and implementation of databases," (To be published).

[12] Wiederhold, G., "Database design (2nd edition)," McGraw-Hill, Inc., 1983.

[13] Wiederhold, G., and Qian, X., "Modeling asynchrony in distributed databases," Invited paper, Proceedings of IEEE Data Engineering Conference, Los Angeles, February 1987.

[14] Chen, P., "The entity-relationship model – toward a unified view of data," from 'Readings in Database Systems' edited by Michael Stonebraker, Morgan Kaufmann Publishers, Inc. 1988.

[15] Hull, R., and King, R., "Semantic data modeling: survey, applications, and research issues," ACM Computing Survey, Vol.19, No.3, September 1987, pp. 243.