Towards a Flexible User Interface to Relational Database Systems: Processing Simple Second Order Queries

Tomasz Imielinski and David Rozenshtein
Department of Computer Science
Rutgers University
New Brunswick, N.J. 08903. U.S.A.

Abstract

In this paper, we present a mechanism for answering simple second order queries of the form "retrieve connection between attributes A and B." By "connection between A and B" we mean *any* and *all* meaningful relationships that can be established between A and B in a given database.

What gives our queries their second order nature is the fact that semantics of connections is independent of a particular set of predicates comprising the database scheme. The answers to them, however, depend on which relationships are considered meaningful; this, in turn, depends on the particular structure of the database scheme and, also, on additional semantic information carried by *roles*. While we can treat our queries as intentionally second order queries directed at the database instance, we can also treat them as incompletely specified queries posed by a user whose knowledge of the database structure is partial.

In this paper, we present a formal characterization of "meaningful relationships," present algorithms for computing connections, and describe how our method can be extended to sets containing any number of attributes.

1. Introduction

In the relational approach, a database scheme can be viewed as a collection of predicates, each over some set of attributes. Equivalently, we can think of a database scheme as a collection of named relation schemes. A query language associated with such a scheme is usually a version of a first order language, such as relational calculus or algebra, generated from the predicates of the scheme.

There are, however, many "real world" situations that give rise to meaningful second order queries: more specifically, the queries, whose internal semantics is independent of any particular set of predicates comprising the scheme. The simple examples of queries of this kind are "Who are the customers of First bank?" or "Who are the students of Professor Smith?"

Note that in neither of the two questions above did we specify any particular relationship between the entities involved. In fact, it is precisely the lack of this specification that gives these two queries their second order semantics, for we are interested in *any* meaningful relationship that exists between customers and banks regardless, for example, of whether a customer has a loan or an account (or anything else, for that matter) with First bank. (We shall explain exactly what we mean by a "meaningful relationship" later in the paper.) Likewise, we are interested in any meaningful relationship between students and professors: for example, Professor Smith can teach a course attended by a student, or he can be an advisor of that student.

Even queries that involve only a single attribute may potentially be second order. For example, consider the query "Who are the employees?" It may be the case that the database scheme does not contain a unary predicate that represents the fact of being an employee. Then, while the intended meaning of the above query remains the same, its evaluation will depend on the particular set of predicates present in the database scheme.

From a formal point of view, the queries of the above type correspond to formulas with quantifies ranging over predicate names of a scheme. We can look at such queries as formulated at some higher level of abstraction, which is, in fact, independent of any particular conceptual scheme.

This type of independence has been studied before. For example, Maier et al [MRSSW,MRW] introduce the notion of *logical data independence*. In particular, they are interested in developing a logically independent query language: one that would enable a user to formulate his queries in a way independent of a particular choice of database predicates. Our approach is different in that, instead of trying to provide logical data independence for all queries, we shall concentrate on those queries which are inherently second order.

Our approach is also different from the one taken by Aho and Ullman [AU]. They are interested in extending the expressive power of relational query languages by incorporating into them certain types of second order operators (such as the least fixed point operator). We, on the other hand, are more interested in associating each second order query with a first order relational expression

that computes that query in a given database scheme. In other words, we want to provide an automatic navigation capability for our queries.

Naturally, we are not attempting to handle arbitrary second order queries; later in the paper, we shall impose considerable restrictions on the class of second order queries that we shall consider.

The remainder of the paper is organized as follows. In the next section we introduce some basic notions, such as scheme, entity, property and role, which will be necessary for future discussions. We also define the class of queries that we shall consider and outline the concept of the answer to such queries. In Section 3 we investigate the notion of "meaningful relationship." In Section 4 we present the algorithms for computing answers to our queries. In Section 5 we discuss the ways of extending our results to handle more complex queries. Section 6 concludes the paper.

2. Basic Notions

We assume a universal set of attributes U. By a database scheme P we consider a set of predicates $\{p_1, ..., p_n\}$, each over some set of attributes X_i. (Each predicate p(X) may be thought of as corresponding to a relation p over scheme X.) We further impose the following restriction on P: $X_i \cap X_j = \phi$ for all $i \neq j$. In other words, any attribute may appear in at most one relation in the database. (Later in the paper we shall remove this restriction.) In addition, we make a natural assumption that every attribute appears in at least one of the relations.

We note that any database can be trivially made to satisfy the above restriction by renaming of attributes (in particular, by prefixing each attribute name with the name of the relation in which it appears). Naturally, both the universal set of attributes U and the set of predicates P are specified by the database designer.

The concepts of particular importance to us are those of *entities* and *roles*. In particular, we need to know which of the attributes represent *entity classes* (all other attributes are considered to represent *properties*). Furthermore, since a given entity may play different roles in different relations in the database, we shall require explicit specification of the role information. We note that both the entity information and the role information are again provided by the database designer. We use the term *extended database scheme* to refer to the database scheme together with this additional information.

The notion of role in an intuitive one and can be best introduced by example. In particular, consider a database with two relations HAS(FACULTY,OFFICE) and TEACHES(INSTRUCTOR,COURSE). We can interpret INSTRUCTOR as the description of the role that some faculty members play with respect to courses. We specify this by saying that attribute INSTRUCTOR is a role of attribute FACULTY, and call (FACULTY,INSTRUCTOR) a *role-pair*.

We note that the notion of role is not a new one and has been used as a tool in describing the world semantics [BD]. By its own nature, the role relationship is a binary relationship among attributes. It is transitive. (For example, if FACULTY is a role of PERSON, and INSTRUCTOR is a role of FACULTY, then INSTRUCTOR is also a role of PERSON). It is antisymmetric. (It is intuitively impossible for two attributes to be roles of each other.) It is not clear, whether the role relationship should be considered reflexive or not. However, the reflexivity does not carry any information about semantic relationships among *distinct* attributes and is, therefore, of no importance to us.

Note that the role relationship carries more semantic information than the *compatibility* relationship [BK], because the latter is always symmetric. Also the role relationship is different from the *subclass* (or *is-a*) relationship [SS,TL]. In particular, consider some database containing attributes A, B and C. Suppose that B has been declared as a subclass of both A and C. Then actdom(B) (standing for the active domain of attribute B, i.e., the set of all values that appear for B anywhere in the database) must be a subset of *both* actdom(A) and actdom(C). For example, if PHYSICAL__CHEMIST is considered to be a subclass of PHYSICIST and also a subclass of CHEMIST, then every physical chemist must be both a physicist and a chemist.

On the other hand, if B has been declared as a role of both A and C, every value in actdom(B) must appear in *at least one of* actdom(A) or actdom(C). For example, if INSTRUCTOR is considered to be a role played by FACULTY and also by ADJUNCT, then every instructor must either be a faculty member or an adjunct.

More formally, the difference between subclasses and roles can be stated as follows. Subclasses correspond to collections of entities grouped according to some set of static properties. In natural language, for example, these properties are usually described by adjectives. Roles, on the other hand, are dynamic. In other words, whether a given entity belongs to a collection of entities described by a given role depends on the relationships (i.e., database scheme predicates) in which the entity "participates." Therefore, in natural language, roles usually correspond to verbs.

Equivalently, we say that subclasses group entities by "what they are," while roles group them by "what they do." For example, given a class of faculty members we can identify a subclass PHD__HOLDER and a role INSTRUCTOR.

The difference between subclasses and roles can be expressed formally using logic. In particular, both subclasses and roles are unary predicates. The difference, however, is that subclass predicates are independent of other predicates in the database scheme. In other words, a subclass predicate is not definable in terms of other predicates in the scheme (except, of course, for the predicate that defines the corresponding class of entities). On the other hand, role predicates are completely definable in terms of other, basic predicates in the scheme. For example, PHD__HOLDER(e) (where e denotes some entity) is not definable in terms of other predicates, while INSTRUCTOR(e) is. In particular, we can define it as INSTRUCTOR(e) $\Leftrightarrow \exists c$ TEACHES(e,c).

In this paper we consider queries whose basic form is "retrieve connection for AB," denoted by [AB], where A

and B are attributes corresponding to entity classes. (The notion of connection has been used before, though in a somewhat different sense [MRW,MW].) Intuitively, [AB] refers to all meaningful relationships between entity classes represented by A and B. We note, that the basic forms of both queries presented in the introduction are indeed connections: for the first query it is [CUSTOMER,BANK], while for the second one it is [STUDENT,PROFESSOR]. In the next section we shall describe what we mean by a "meaningful relationship" in detail. In general, however, we can think of "connection" as a metapredicate whose interpretation varies depending on the extended database scheme.

What is important in the above definition is that we are not looking for connections between pairs of attributes in which one of them does not represent an entity class. Indeed, we believe, that there is no sense in asking about connections between two properties or an entity class and a property. In the latter case, if a given property is declared as a property of a given entity class, then the connection exists; if not, then it doesn't.

There are two possible concepts of the *answer* for queries of the form described above. One one hand, we can treat such queries *literally* as second order queries addressed at the *database instance*. On the other hand, we can treat such queries as *incompletely specified* queries posed by a user whose knowledge of the database scheme is partial.

In the first case, the answer is simply a set of all tuples comprising the corresponding connection. In the second case, the query is really addressed at the *database scheme*: the answer to such a query should then be defined as the set of all relational algebra expressions that would participate in the computation of the corresponding connection.

We note that the second interpretation allows us to accommodate those users who do not have complete knowledge of the database structure. This situation is different from those analyzed as part of research on incomplete databases (Lipski [Li] provides an overview of this research) where it is the database that does not have complete knowledge of the world.

We consider both interpretations natural and worthy of further investigation. Before that, however, we shall clarify what we really mean by a "meaningful relationship."

3. Meaningful Relationships

We begin this section by presenting several example queries and database schemes and describe those relationships that we consider to be meaningful. For the sake of clarity, we shall use a hypergraph notation to pictorially represent database schemes. In other words, we shall denote a relation scheme p(X) by listing the attributes in X and then enclosing exactly them within a solid line. (Thus, in a hypergraph representation of a database scheme, each hyperedge will correspond to a relation scheme.) Note, that because of our restriction, there may be at most one predicate over any given set of attributes. Therefore, we do not need the predicate names to identify the predicates, and, thus, we can omit the predicate names from the hypergraphs. As a matter of notation, we shall use p(X) to denote a relation corresponding to the predicate over the set X (if it exists, of course).

Suppose that a user has a database that contains various information about a university. Furthermore, assume that his knowledge of the database structure is incomplete. In particular, he knows that the database contains attributes FACULTY and STUDENT, which represent two entity classes: class of faculty members and class of students, but doesn't know precisely what the relationship between faculty members and students is. Therefore, the user is interested in establishing a connection [FACULTY,STUDENT] between these attributes. In other words, he is interested in finding any and all meaningful relationships between FACULTY and STUDENT.

Suppose that the scheme of the university database consists of a single predicate as presented in Figure 1.

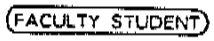

Figure 1

It is obvious that there exists a meaningful relationship between FACULTY and STUDENT. Furthermore, this relationship is explicitly represented by the predicate p(FACULTY,STUDENT). Therefore, we return this predicate as the answer to the user's request. (Alternatively, we say that we define connection [FACULTY,STUDENT] to be p(FACULTY,STUDENT).)

Suppose, however, that the database scheme was changed to consist of a single predicate of the form p(FACULTY,COURSE,STUDENT) as presented in Figure 2. It is again obvious that in this database we can define connection [FACULTY,STUDENT] to be the projection of this predicate onto {FACULTY,STUDENT}, $\pi_{FS}(p(FCS))$. (From now on we shall abbreviate attribute names by their first letters.)

Figure 2

Suppose, however, that the we received additional information; in particular, we learned that faculty members instruct students and research various topics. Therefore, we can modify the database scheme accordingly. The new database scheme is presented in Figure 3. In it p(F) lists all faculty members.

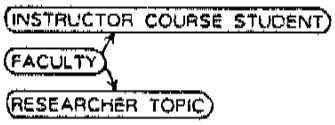

Figure 3

For the sake of succinctness, we shall represent role information pictorially directly on the hypergraph. In particular, we shall draw an arrow (called *role-arrow*) from attribute A to attribute B if B is specified as a role of A. In Figure 3, for example, INSTRUCTOR is a role of FACULTY; likewise, RESEARCHER is a role of FACULTY.

April, 1984

TOWARDS A FLEXIBLE USER INTERFACE TO RELATIONAL DATABASE SYSTEMS: PROCESSING SIMPLE SECOND ORDER QUERIES

T. Imielinski and D. Rozenshtein

DCS-TR-142

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

From the semantic point of view, there exists exactly the same relationship between faculty members and students in this database as in the one of Figure 2. In particular, faculty member Smith is related to student Frye if and only if predicate p(ICS) contains a tuple <Smith,c,Frye>, where c is some course. Therefore, in this database, we define [FS] as

$$\pi_{FS}(p(F) \overset{F=I}{\bowtie} p(ICS)).$$

Consider further refining our database. In particular, suppose that we now know that, in addition to teaching students, faculty members also advise them. The revised database scheme is presented in Figure 4.

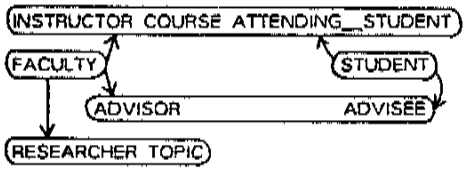

Figure 4

Note that the introduction of new names for attribute STUDENT is mostly due to our restriction that each attribute may appear in only one relation.

In this database, there exist two semantically distinct, yet perfectly meaningful relationships between faculty members and students. On one hand, faculty members instruct students. This relationship may be established as

$$\pi_{FS}(p(F) \overset{F=I}{\bowtie} p(ICAt) \overset{At=S}{\bowtie} p(S)),$$

where At stands for ATTENDING__STUDENT. On the other hand, faculty members advise students. This relationship may be established as

$$\pi_{FS}(p(F) \overset{F=Ar}{\bowtie} p(ArAe) \overset{Ae=S}{\bowtie} p(S)),$$

where Ar stands for ADVISOR and Ae stands for ADVISEE. Therefore, both of these relationships should be reflected in the definition of [FS].

We note that it really does not matter to us which of the two possible concepts of the answer (discussed at the end of the previous section) is chosen. If the first concept is chosen, then we shall return a union of the results of the above two expressions. If the second one is chosen, then we shall return the two expressions themselves. What is important, however, is that *both* of these relationships participate in the definition of connection between FACULTY and STUDENT.

Note that the intended meaning of the query "retrieve connection between FACULTY and STUDENT" remained the same in all of the above examples. What changed, however, was the database scheme. That, in turn, changed how [FS] was defined.

Consider further refining our database. In particular, suppose that some of the topics become basis for research proposals submitted to various agencies, and the user is now interested in the connection between faculty members and agencies. The revised database scheme is presented in Figure 5.

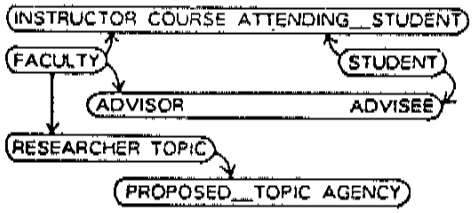

Figure 5

In this database, we can establish a meaningful relationship between FACULTY and AGENCY as

$$\pi_{FAg}(p(F) \overset{F=R}{\bowtie} p(RT) \overset{T=P}{\bowtie} p(PAg)),$$

where Ag stands for AGENCY. A tuple <f,a> from the result of this expression is interpreted as "faculty member f is a researcher studying some topic on which he wrote a proposal to agency a." (As an additional argument, consider a semantically equivalent database not satisfying our restriction. For example, such database may consist of two relations: RESEARCHES(FACULTY,TOPIC) and PROPOSED(TOPIC,AGENCY). It is intuitively correct to establish the connection between FACULTY and AGENCY by a natural join of the two relations. This natural join corresponds directly to the equijoin taken above.)

We can use the above examples to specify what we really mean by a "meaningful relationship." In particular, we say that there exists a *meaningful relationship* between attributes A and B (both A and B represent entity classes) if the database scheme contains a predicate p(X), called *connection predicate*, such that one of the following conditions holds.

1. Both attributes A and B directly participate in p(X); in other words, both A and B are elements of X.
2. Attribute A plays some role in p(X), while B directly participates in p(X).
3. Both attributes A and B play some roles in p(X).

We can specify what we mean by "A plays some role in p(X)" further. In particular, we consider the following two cases:

(i) Attribute A plays a *direct role* in p(X). In other words, there exists some attribute C, such that C is a role of A and C ∈ X).

(ii) Attribute A doesn't play a direct role in p(X), yet there exists a path from attribute A to predicate p. (We shall specify the nature of this path later in the paper.)

The database schemes presented in Figure 1 and Figure 2 satisfy the first condition with respect to attributes FACULTY and STUDENT. The database scheme of Figure 3

satisfies the second condition. In particular, it contains predicate p(ICS), such that FACULTY plays the direct role of INSTRUCTOR in p(ICS) and STUDENT directly participates in p(ICS).

The database scheme of Figure 5 also satisfies the second condition. It contains predicate p(PAg), such that AGENCY directly participates in p(PAg) and we can establish a path from FACULTY to p(PAg). In particular, the path is (1) from attribute FACULTY *along* a role-arrow to attribute RESEARCHER; (2) from attribute RESEARCHER to attribute TOPIC using predicate p(RT); and finally (3) from attribute TOPIC along a role-arrow to attribute PROPOSED_TOPIC, and thus to predicate p(PAg).

Finally, the database scheme of Figure 4 satisfies the third condition. In fact, it contains two predicates: p(ICAt) and p(ArAe), in which FACULTY and STUDENT play some roles.

We can use the same university database examples to identify certain relationships which we do not consider meaningful. More precisely, we do not consider them meaningful with respect to our problem of defining answers to connections.

Consider again the database scheme of Figure 3. Does there exist a meaningful (in our sense) relationship between FACULTY and INSTRUCTOR in this database? We note that the only way in which a relationship between FACULTY and INSTRUCTOR can be established is by taking the equijoin between p(F) and p(ICS), followed by a projection onto FI. In other words, one might try to define [FI] as

$$\pi_{FI}(p(F) \overset{F=I}{\bowtie} p(ICS)).$$

We think, however, that this approach is incorrect. Indeed, such a join, followed by the projection, would result in a set of tuples of the form <f,f>, thus providing redundant information. On the other hand, the information contained in such a result would be in some sense incomplete, since any tuple <f,f> from this relation will have to be interpreted as "f is a faculty member, which also plays a role of instructor;" yet it is not clear what this role is with respect to. (Note that in the case of *subclass* relationships such interpretations are semantically complete. Therefore, binary relations of this form do make sense there.)

In addition, we think that if the user really wanted to know which entities play a role of instructor, he would have asked for the connection for INSTRUCTOR alone. We shall show how single attribute connections are computed later in this paper.

Therefore, we can refine our notion of meaningful relationship by requiring that neither of the attributes, for which the relationship is being established, be used as a *role indicator* in establishing the relationship.

Consider the database scheme presented in Figure 6. It is a modification of the scheme of Figure 3. In particular, a new class of entities, denoted by ADJUNCT, who can also teach courses, has been added to the database, while the information about researches and their research topics has been deleted.

Figure 6

Does there exist a meaningful relationship between FACULTY and ADJUNCT in this database? The only way one might try to establish this relationship is by computing

$$\pi_{FA}(r(F) \overset{F=I}{\bowtie} r(ICS) \overset{I=A}{\bowtie} r(A)).$$

We reject this approach as semantically meaningless, since the above expression would again always result in a set of tuples of the form <f,f>. In addition, the class of (full time) faculty members and the class of (part time) adjuncts are usually disjoint; thus, the result of the above equijoin would probably be the empty relation.

This example allows us to further refine our notion of meaningful relationship. In particular, we require that the attributes (in the connection predicate) used as role indicators be distinct.

As a final example of relationships, which we do not consider to be meaningful, consider the database scheme of Figure 7. (The intended meaning of p(FR) is that faculty members use rooms.)

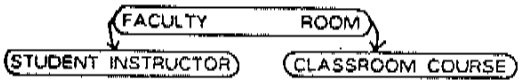

Figure 7

In this database, there does not exist a way to establish a meaningful relationship between INSTRUCTOR and CLASSROOM; indeed, from knowing that faculty member Smith uses room #1 (and even from knowing that Smith is an instructor and room #1 is sometimes used as a classroom), we cannot conclude that Smith as *instructor* uses room #1 as a *classroom*. In fact, it is entirely possible that Smith uses that room as an office (or anything else, for that matter), while somebody else uses it as a classroom.

We can also make the following, more general argument. Recall that the user's knowledge of the database structure may be partial. Therefore, his queries should be interpreted as incompletely specified. Consider some such query that involves an attribute A. Suppose the database scheme is such that it contains some attributes which are roles of A; furthermore, it contains attributes for which A is a role.

Since the user's knowledge of the scheme is incomplete, we may assume that, in posing the query, he was unaware of any role that entities from A may play; on the other hand, we must assume that his use of A was conscious. In other words, we must assume that he is indeed interested only in entities that play the role denoted by A, and not in entities that play any of the more general roles. Therefore, in answering the user's query (i.e., in establishing the corresponding connection) we may use any of A's

roles, but may not use any of the attributes more general than A.

Consider again the database of Figure 7. The only candidate for connection predicate is p(FR). The only path from attribute INSTRUCTOR to this predicate is through attribute FACULTY. But FACULTY is not a role of INSTRUCTOR; in fact, FACULTY is more general than INSTRUCTOR. Likewise, ROOM is more general than CLASSROOM.

This example allows us to refine our notion of meaningful relationship even further. Recall that establishing a meaningful relationship between two attributes A and B basically reduces to finding some connection predicate p, such that we can construct paths from A to p and from B to p.

In constructing a path from A to p, we want to preserve the role that A assigns to its entities; therefore, we can only use A's roles. Pictorially, that means that the first step in the path must be *along* a role-arrow from A to some attribute C. If C is included in p, then we are done. (Recall that in this case we say that A plays a direct role in p.) Alternatively, C must be an attribute that appears in some predicate q(Y). We can try to establish a path from q to p.

In order to do this, we must find some attribute D ∈ Y, such that we can construct a path from D to p. However, in constructing the path from D to p, we must now also preserve the role that D assigns to its entities. Therefore, we may leave q only *along* some role-arrow that originates at D. If this last role-arrow brings us to p, we are done; if not, we can try to extend the path further. In doing so, however, we must always preserve the roles of all the attributes already introduced into the path. In other words, in extending the path, we may only move along role-arrows.

4. Formalization of Connections

We now present a general rule on how to establish connections between any pair of attributes, which represent entity classes. This rule is motivated by the examples and discussion presented in the previous section.

Recall that there are two possible interpretations of what an answer to a connection should be. One one hand, the answer to a connection is the set of all relational algebra expressions which represent various meaningful relationships between a given pair of attributes. On the other hand, it is the union of the results of these expressions. For the sake of simplicity, in this paper, we shall adopt the second interpretation.

Consider establishing connection between two attributes A and B. Recall that P is the set of predicates (or relation schemes) $p(X_i)$. If P contains some predicate $p(X)$, such that both A and B directly participate in p (i.e., A ∈ X and B ∈ X), then we can trivially establish a meaningful relationship between A and B simply by projecting p onto AB.

On the other hand, suppose that only one of the attributes, say B, directly participates in p. We can still establish a meaningful relationship between A and B, if we can find a role preserving (i.e., satisfying all of the restrictions discussed at the end of the previous section) path from A to p.

Formally, we say that there exists a *role preserving* path from attribute A to predicate p through role indicator F if the following condition holds. There exists a sequence of predicates $p(X_0)$, ..., $p(X_n)$ and a sequence of *distinct* role-pairs (C_0,D_1), ..., (C_{n-1},D_n) for some $n > 0$, such that

(a) $C_i \in X_i$ for $i = 0, ..., n-1$;

(b) $D_i \in X_i$ for $i = 1, ..., n$;

(c) $A = C_0$;

(d) $F = D_n$;

(e) $p(X) = p(X_n)$ (i.e., $X = X_n$).

In other words, we start from attribute A, use the role-arrow from A to D_1, use $p(X_1)$ to get from D_1 to C_1, use the role-arrow from C_1 to D_2, etc., and finally enter p through attribute F.

We note that we do not want to consider infinite paths. Therefore, we must restrict ourselves to either sequences of distinct predicates or sequences of distinct role-pairs. We choose the latter, because there exist certain cases in which we do want to use the same predicate more that once. In particular, consider establishing a connection between faculty members and courses in the database, whose scheme is presented in Figure 8. The intended meaning of p(FSC) is that faculty members teach courses to students. However, some faculty members sit in on the courses taught by others; therefore, in this case, they play the role of students.

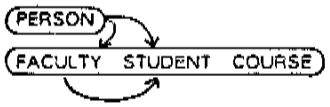

Figure 8

Naturally, since faculty members teach courses, $\pi_{FC}(r(FSC))$ represents a meaningful relationship between FACULTY and COURSE. On the other hand, faculty members take courses. We can establish this relationship by considering a non-empty path from FACULTY to p(FSC). In particular, this path would be: from attribute FACULTY in p(FSC) along a role-arrow to attribute STUDENT in p(FSC), thus using predicate p(FSC) twice.

If indeed there exists a role preserving path from attribute A to predicate p(X) through attribute F, $B \in X$ and $F \neq B$, then we can compute a meaningful relationship between A and B as follows.

Algorithm 1

1. Compute $s_n \leftarrow \pi_{FB}(p(X))$.
   (Note that $F = D_n$.)

2. For i from n - 1 step 1 downto 1 do
   Compute $s_i \leftarrow \pi_{D_iB}(\pi_{D_iC_i}(p(X_i)) \overset{C_i=D_{i+1}}{\bowtie} s_{i+1})$.
   (Note that s_i has D_iB as its scheme.)

3. Compute $s_0 \leftarrow \pi_{C_0B}(\pi_{C_0}(p(X_0)) \overset{C_0=D_1}{\bowtie} s_1)$.
   Return s_0 as the result.
   (Note that $A = C_0$;
   thus, the scheme of s_0 is AB.)

Note that relations $p(X_1)$ through $p(X_{n-1})$ are needed only for the purpose of computing the equijoin. Furthermore, during the computation, only attributes D_i and C_i from relation $p(X_i)$ are used. Hence, the projection of relation $p(X_i)$ onto D_iC_i in Step 2.

Now consider the case when neither attribute A nor attribute B directly participate in p. As before, we can still establish a meaningful relationship between A and B, if we can find two role preserving paths: one from A to p through attribute F, and the other from B to p through attribute G, such that $F \neq G$. In there exist two such paths, then we can establish a meaningful relationship between A and B as follows.

Algorithm 2

1. Compute the relationship between A and G using Algorithm 1. Call resulting relation r.

2. Compute the relationship between B and A using Algorithm 1. (use r in place of p(X) in Step 1 of the algorithm).

We illustrate the above algorithms by considering various connections in the following person-bank database.

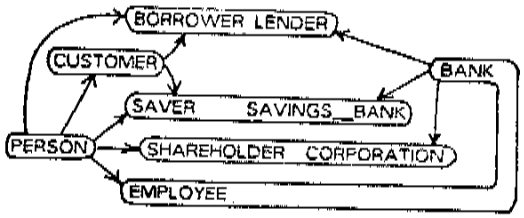

Figure 9

The intended meaning of relations in this database is that borrowers borrow money from lenders; savers save money at savings banks; shareholders hold shares of stock in corporations; and, finally, banks have employees.

Consider the following user query: "What is the connection between customers and banks?" Our algorithms will define [CuBa], where Cu stands for CUSTOMER and Ba stands for BANK, as (we simplified the expression)

$$\pi_{CuBa}(p(Cu) \overset{Cu=Bo}{\bowtie} p(BoL) \overset{L=Ba}{\bowtie} \pi_{Ba}(p(EBa))) \cup$$

$$\pi_{CuBa}(p(Cu) \overset{Cu=Sa}{\bowtie} p(SaSb) \overset{Sb=Ba}{\bowtie} \pi_{Ba}p(EBa)),$$

which is intuitively correct. (In the expression above, Bo stands for BORROWER, Sa stands for SAVER and Sb stands for SAVINGS_BANK.)

As another example, consider the query "What is the relationship between people and banks?" (Note that this query is in some sense less specified than the previous one; in particular, the user is interested in any role (and not just in those consistent with the role of customer) that people may play with respect to banks.) Our algorithms will define [PBa] as the following union (we again simplified the expression):

$$\pi_{PBa}(p(P) \overset{P=Bo}{\bowtie} p(BoL) \overset{L=Ba}{\bowtie} \pi_{Ba}(p(EBa))) \cup$$

$$\pi_{PBa}(p(P) \overset{P=Sa}{\bowtie} p(SaSb) \overset{Sb=Ba}{\bowtie} \pi_{Ba}(p(EBa))) \cup$$

$$\pi_{PBa}(p(P) \overset{P=Sh}{\bowtie} p(ShCo) \overset{Co=Ba}{\bowtie} \pi_{Ba}(p(EBa))) \cup$$

$$\pi_{PBa}(p(P) \overset{P=E}{\bowtie} p(EBa)),$$

which is again intuitively correct. (In the expression above, Sh stands for SHAREHOLDER and Co stands for CORPORATION.)

We can also establish other connections in the database of Figure 9. For example, the connection between PERSON and SHAREHOLDER would be defined as (we simplified the expression)

$$\pi_{PSh}(p(P) \overset{P=E}{\bowtie} p(EBa) \overset{Ba=Co}{\bowtie} p(CoSh)).$$

A tuple <Smith,Brown> from this connection is interpreted as "person Smith works for some bank whose shares of stock are owned by Brown."

We now finally pass to the general case and consider databases with no restrictions imposed on the database scheme. In other words, any attribute may now occur in several different relation schemes. The importance of considering the restricted case first lies in that we now have a proper framework to deal with the unrestricted one. Furthermore, we can now provide an intuitive and simple correction criteria for any connection definition mechanism.

Recall, that any database can be made to satisfy our restriction simply by renaming different occurrences of the same attribute A in different schemes. Naturally, we want this transformation to be information preserving. In other words, we want to preserve the meaning of all connections that existed in the original, unrestricted database. Therefore, a connection definition mechanism

may be considered correct only if the result of every connection taken in the unrestricted database remains the same when the connection is taken in the new, renamed database. (We shall, of course, have additional connections, between the new attributes, in the renamed database.)

It is the case, that our algorithms are correct (in the above sense) if the transformations indeed preserve the meaning of the database scheme.

In particular, consider retrieving a connection between employees and departments in the database whose scheme is presented in Figure 10. The intended meaning is that employees work in departments and managers manage departments.

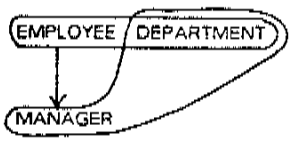

Figure 10

From an intuitive point of view, since employees work in departments, p(ED) should certainly be returned as part of the connection. In addition, since employees also manage departments,

$$\pi_{ED}(\pi_{E}(p(ED)) \overset{E=M}{\bowtie} p(MD))$$

should be returned. Note that our algorithms will return precisely the union of the two expressions above.

As an additional argument, consider constructing a semantically equivalent database scheme that would satisfy our previous restriction. In designing databases, we subscribe to the intuitive *principle of semantic minimality* [Ro]. Basically, this principle states that any relationship is explicitly represented in the database scheme only once. In particular, if the database contains two relations r(X) and s(Y) and there exist attributes A, B and C such that $A \in X$, $B \in Y$, $C \in X \cap Y$, and B is a role of A, then the meaning assigned by relation r to the relationship between A and C is *semantically different* from the meaning assigned by relation s to the relationship between B and C.

In our case, the principle of semantic minimality would require that the semantics carried by p(ED) be different from the semantics carried by p(MD). It is obvious that this is, in fact, so. Furthermore, we cannot assume that $p(MD) \subseteq p(ED)$, since, in general, a person may be a manager of a department without necessarily being an employee of the *same* department.

Since we cannot assume any specific relationship between the occurrence of attribute DEPARTMENT in relation p(ED) and its occurrence in relation p(MD), we rename it in both places and introduce a new relation over just attribute DEPARTMENT itself. Furthermore, we consider the two new attributes to be roles of DEPARTMENT (one with respect to employees and the other with respect to managers). This new, renamed version of the database scheme is presented in Figure 11. The intended meaning of p(D) is to list all existing departments.

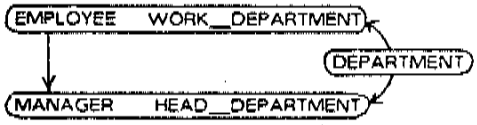

Figure 11

Since this database scheme is semantically equivalent to the one presented in Figure 10, the meaning of any connection computed in the previous database should remain the same. In particular, [ED], as computed for the database scheme of Figure 10, should be equal to [ED], as computed for the scheme above. We note that this is indeed so, since our algorithms would compute the connection between employees and departments as

$$\pi_{ED}(p(EW) \overset{W=D}{\bowtie} p(D)) \cup \pi_{ED}(\pi_{E}(p(EW)) \overset{E=M}{\bowtie} p(MH) \overset{H=D}{\bowtie} p(D)).$$

What happens, however, if the attribute that is mentioned in more than one relation has roles of its own? For example, consider the database scheme of Figure 12. The intended meaning is that some corporations are public and thus have shareholders, and some corporations are private and thus have owners. Furthermore, all corporations are either public or private. In addition, some corporations design hotels.

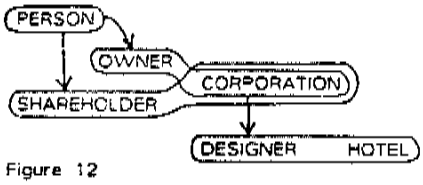

Figure 12

Consider establishing a connection between persons and hotels. Our algorithms will return

$$\pi_{PH}(p(P) \overset{P=O}{\bowtie} p(OC) \overset{C=D}{\bowtie} p(DH)) \cup$$

$$\pi_{PH}(p(P) \overset{P=S}{\bowtie} p(SC) \overset{C=D}{\bowtie} p(DH))$$

as the answer to [PH], which is intuitively correct. Consider, however, renaming attribute CORPORATION in relations p(OC) and p(SC). Since, again, we cannot assume any specific semantic relationship between the occurrence of attribute CORPORATION in p(OC) and its occurrence in p(SC), we rename CORPORATION in both places (say, to PRIVATE__CORPORATION and PUBLIC__CORPORATION) and introduce p(C), listing all corporations, into the database scheme. Furthermore, we specify that both new attributes are roles of attribute CORPORATION.

We note that the role relationship between CORPORATION and DESIGNER remains. However, if we want this transformation to be information preserving (in particular, if we want [PH] to be computed in the same way as above), we must add two new role-pairs to the database scheme; namely, DESIGNER is a role of

PRIVATE__CORPORATION and also the role of PUBLIC__CORPORATION. The database scheme corresponding to this transformation is presented in Figure 13.

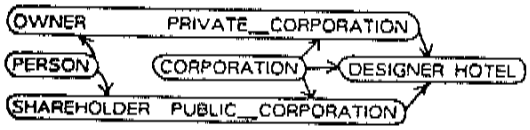

Figure 13

We note, that any connection, which could be computed in the database of Figure 12, would be computed to be exactly the same in this database.

5. Extensions

It is quite easy to extend our mechanism to handle connections for single attributes. In particular, consider establishing a connection [A] for some attribute A. If the database satisfies our restriction, then A will appear in exactly one predicate p. In this case, we define [A] to simply be $\pi_A(p)$. Alternatively, let A appear in the set of predicates $p_1, \ldots p_n$. Then we define [A] as the following union:

$$\pi_A(p_1) \cup \ldots \cup \pi_A(p_n).$$

It is also straightforward to extend our mechanism to handle queries that involve selections. For example, consider again the query "Who are the customers of the First bank?" addressed at the database of Figure 9. We answer this query by first identifying the underlying connection (in this case, between customers and banks) and then by applying the selection condition "BANK = First" to the result of this connection.

We note that, in general, once the user obtains the (set of) connections he is interested in, he can use any relational algebra operations to combine the information contained in them.

Finally, we believe that our mechanism can be extended in a natural way to handle connections for sets containing more than two attributes. Consider a user of a (probably police) database, which contains various information about people and guns. The user is interested in finding out whether anything happened on January 31, 1983 that involved person Smith and the gun Colt, serial number 12345. The database scheme is presented in Figure 14.

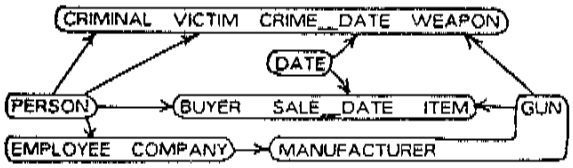

Figure 14

In order to answer this question, we first have to establish connection among persons, dates and guns. From an intuitive point of view, this database contains three, semantically distinct, but meaningful relationships among PERSON, DATE and GUN. In particular, the first of these relationships may be established as

$$\pi_{PDG}(p(P) \overset{P=Cr}{\bowtie} p(CrVCdW) \overset{W=G}{\bowtie} p(G) \overset{Cd=D}{\bowtie} p(D)),$$

where Cr stands for CRIMINAL and Cd stands for CRIME__DATE. A tuple <p,d,g> from the result of this expression would be interpreted as "person p committed a crime on date d with gun g."

The second of these relationships may be established as

$$\pi_{PDG}(p(P) \overset{P=V}{\bowtie} p(CrVCdW) \overset{W=G}{\bowtie} p(G) \overset{Cd=D}{\bowtie} p(D)),$$

A tuple <p,d,g> from the result of this expression would be interpreted as "person p was a victim of a crime committed on date d with gun g."

And finally, the third of these relationships may be established as

$$\pi_{PDG}(p(P) \overset{P=B}{\bowtie} p(BSI) \overset{I=G}{\bowtie} p(G) \overset{S=D}{\bowtie} p(D)).$$

A tuple <p,d,g> from the result of this expression would be interpreted as "person p bought gun g on date d."

Note that in each of the cases above, there existed some connection predicate to which we could find a role preserving path from each of the three attributes in question. Furthermore, the attributes through which these paths entered the connection predicate were distinct. We can generalize this observation as follows. Given an extended database scheme and a set of attributes $\{A_1, \ldots A_n\}$, there exists a meaningful relationship among the attributes in the set if the database scheme contains a connection predicate p, such that for each A_i there exists a role preserving path from A_i to p through attribute F_i with all F_i being distinct. We then define connection $[A_1,\ldots,A_n]$ among these attributes to be a combination of all meaningful relationships that exist among them.

6. Conclusions

In this paper we consider a simple class of second order queries, each of the form "retrieve connection between A and B," where A and B are attributes representing classes of entities. By "connection between A and B" we mean *any* and *all* "meaningful relationships" between A and B.

What gives our queries their second order semantics is the fact that the intended meaning of a query is independent of a particular set of predicates (relation schemes) comprising the database scheme. On the other hand, the evaluation of a query (and, therefore, the precise meaning of the answer) will depend on which relationships between the two attributes involved are considered meaningful. This, in turn, will depend on the particular structure of the database scheme.

In deciding which relationships between attributes are meaningful, we make use of additional semantic information; in particular, we need to know which

attributes represent *entity classes* and what are the *role relationships* among attributes. In Section 2 of this paper, we discuss the notion of *role* in detail. In Section 3 we present several examples motivating the formal definition of "meaningful relationship" presented in Section 4. In Section 4 we also present algorithms for computing these relationships.

In Section 5 we outline how our mechanism for defining connections between pairs of attributes can be extended to include selection (and other relational algebra operations); we also describe how it can be extended to sets containing any number of attributes.

Recall that our queries can be treated as incompletely specified queries posed by a user whose knowledge of the database scheme is partial. We believe that there is much research to be done to handle such situations, where, as most of our examples illustrate, the user's queries are inherently second order.

The process of answering such queries not only increases the user's knowledge of the "real world" (which is the main purpose of a database system) but also completes his knowledge of database structure (or scheme). Depending on the initial state of the user's knowledge of the database scheme, we can think of a number of strategies for increasing this knowledge during the query answering process.

In this paper, we addressed one particular situation where the user's knowledge of the database scheme is limited to just attributes; during the query answering process, the user acquires knowledge about relationships.

This direction of research is particularly important if we want to provide interfaces friendly to so called "naive" users. In particular, it should not necessary for a user to know the whole structure of the database scheme before starting his interaction with the database. Instead, the interface should hide the usually enormous complexity of the database scheme and provide such knowledge only upon request.

Bibliography

[AU] A.V. Aho, J.D. Ullman. Universality of Data Retrieval Languages. *ACM Symposium on Principles of Programming Languages*, 1979

[BD] C.W. Bachman, M. Daya. The Role Concept in Data Models. *Proceedings of the 3rd International Conference on Very Large Databases*, 1977.

[BK] C. Beeri, H.F. Korth. Compatible Attributes in a Universal Relation. *Proceedings of the ACM Symposium on Principles of Database Systems*, March 1982.

[Co] E.F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4:4, pp. 397–434, December 1979.

[Ho1] P. Honeyman. Extension Joins. *Proceedings of the 6th Conference on Very Large Data Bases*, October 1980.

[Ho2] P. Honeyman. Testing Satisfaction of Functional Dependencies. *Journal of the ACM*, 29:3, pp. 668 – 677, July 1982.

[IL] T. Imielinski, W. Lipski. A Systematic Approach to Relational Database Theory. *Proceedings of the ACM-SIGMOD Conference*, June 1982, Orlando, Fla.

[KU] H.F. Korth, J.D. Ullman. System/U: A Database System Based on the Universal Relation Assumption. *Proceedings of the XP1 Workshop on Relational Database Theory*, June–July 1980, Stony Brook, N.Y.

[Li] W. Lipski. Logical Problems Related to Incomplete Information in Databases. Presented at *Colloquium on Algebra, Combinatorics and Logic in Computer Science*. September 1983, Gyor, Hungary.

[Ma] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[MRSSW] D. Maier, D. Rozenshtein, S. Salveter, J. Stein, D.S. Warren. Towards Logical Data Independence: A Relational Query Language without Relations. *Proceedings of the ACM-SIGMOD Conference*, June 1982, Orlando, Fla.

[MRS] D. Maier, D. Rozenshtein, J. Stein. Representing Roles in Universal Scheme Interfaces. To appear in *Proceedings of the IEEE Computer Data Engineering Conference*, April 1984, Los Angeles, Ca.

[MRW] D. Maier, D. Rozenshtein, D.S. Warren. Windows on the World. *Proceedings of the ACM-SIGMOD Conference*, May 1983, San Jose, Ca.

[MW] D. Maier, D.S. Warren. Specifying Connections for a Universal Relation Database Scheme. *Proceedings of the ACM-SIGMOD Conference*, June 1982, Orlando, Fla.

[MU] D. Maier, J.D. Ullman. Maximal Objects and the Semantics of Universal Relation Databases. *ACM Transactions on Database Systems*, 8:1, pp. 1–14, March 1983.

[Ro] D. Rozenshtein. *Query and Role Playing in the Association-Object Data Model*. Doctoral Dissertation, State University of New York at Stony Brook, 1983.

[Sa] Y. Sagiv. Can We Use the Universal Instance Assumption without Using Nulls? *Proceedings of the ACM-SIGMOD Conference*, April – May 1981, Ann Arbor, Mich.

[Sc] E. Sciore. *The Universal Instance and the Database Design*. Doctoral Dissertation, Princeton University, 1980.

[SS] J.M. Smith, D.C.P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems*, 2:2, pp. 105–133, June 1977.

[TL] D. C. Tsichritzis, F.H. Lochovsky. *Data Models*. Prentice–Hall, 1982.

[Ul] J.D. Ullman. *Principles of Database Systems*. Computer Science Press, 1980.

[Ya] M. Yannakakis. Algorithms for Acyclic Database Schemes. *Proceedings of the 7th Conference on Very Large Data Bases*, September 1981.