Using Integrity Constraints to Provide Intensional Answers to Relational Queries

Amihai Motro

Computer Science Department
University of Southern California
University Park, Los Angeles, CA 90089-0782

Abstract

An *intensional* answer to a query is a set of characterizations of the set of database values that satisfy the query (the *extensional* answer). Intensional answers provide users with additional insight into the nature of standard extensional answers. In this paper we describe a method that applies database *constraints* to generate intensional answers. These intensional answers characterize the extensional answers in two ways: (1) with constraints that are applicable to the extensional answer, and (2) with database views that are contained entirely in the extensional answer. Our method is to represent the definitions of constraints in special "meta-relations", and extend standard algebraic operators to these relations. When a query is presented to the database system, it is performed *both* on the actual relations, resulting in an extensional answer, and on the meta-relations, resulting in definitions of constraints that apply to the extensional answer, as well as database views that are contained entirely in the extensional answer. These definitions are translated into an intensional answer that accompanies the extensional answer.

1 Introduction

The information stored in databases is of two kinds. *Extensional* information (often called *data*) is information that applies to individual real world objects. *Intensional* information (often called *knowledge*) is information that applies to multitudes of real world objects. In the relational database model extensional information is expressed with *relations* over domains of *data values*, and intensional information is expressed with *constraints*, which are formulas in predicate logic that assert required relationships among the data values.

Relational database systems use constraints to enhance the integrity of the extensional information in the database by monitoring changes made to the relations. Assuming that initially the relations satisfy the constraints, thereafter update requests are accepted only if the resulting relations would not violate any of the constraints. For reasons of efficiency, most database systems do not implement constraints that are arbitrary predicate logic formulas. Usually, only very specific types of formulas are allowed, such as *range* constraints or *referential* constraints. But even a database system that implements only the notion of *relation key*, forbidding two tuples from having the same key value, is, in effect, enforcing a particular kind of constraint.

An *answer* to a query is a set of data values that satisfy a qualification specified in the query. Therefore, answers are derived entirely from the extensional information in the database. Indeed, the only intensional information that characterizes this set of values is the qualification specified in the query that generated it. Still, the intensional information in the database may include additional characterizations of the extensional answer. If this intensional information is retrieved, database answers may gain additional meaning.

In this paper we describe a method that provides such intensional answers. As an example, assume a personnel database that includes a relation EMPLOYEE with attributes NAME, TITLE, SALARY and DEPARTMENT, and two constraints: one states that all employees in the

This work was supported in part by NSF Grant No. IRI-8609912 and by an Amoco Foundation Engineering Faculty Grant.

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

design department are guaranteed a salary of at least $30,000, and the other states that all employees in research positions are in the design department. A query to retrieve all the employees in the design department may generate the extensional answer:

- Betty, Harry, Mary, Tom

and the intensional answer:

- All employees retrieved earn at least $30,000.
- All employees in researcher positions retrieved.

Thus, an intensional answer characterizes the extensional answer, providing more insight into the nature of the set of values retrieved.

In our approach, providing intensional answers from constraints is a specific case of the following more general problem: Given a query and set of database views that possess a particular property, what views of the answer possess this property?

In [6] we considered database views whose property is that they have guaranteed *integrity*. The problem then became: Given a query, what views of its answer have guaranteed integrity? Solving this problem enabled us to extend a database system so that every answer is accompanied by statements that define its integrity, resembling a *certification of quality*. In [5] we considered database views whose property is that they are *permitted* to a particular user. The problem then became: Given a query, what views of its answer should be permitted to this user? Solving this problem enabled us to extend a database system with an *authorization mechanism* that has several advantages over other known mechanisms.

In this paper we consider database views whose property is that they are always *null*. The problem then becomes: Given a query, what are the views of its answer that are always null? As we shall see, views of this type express constraints, and thus provide good intensional characterizations of answers.

In each case, our solution is to represent the definitions of the given database views in special relations, using the concept of *meta-tuples*. A meta-tuple defines a subview (i.e., a selection and a projection) of a single relation, and several meta-tuples can be used together to define general views. All meta-tuples that define subviews of the same relation are stored together in one *meta-relation*, whose structure mirrors the actual relation. Standard algebraic operators (product, selection and projection) are extended to these meta-relations.

When a query is presented to the database system, it is performed *both* on the actual relations, resulting in an answer, and on the meta-relations, resulting in definitions of views of the answer that inherit the particular property of the given views.

In the case of constraints, these views express constraints that apply to the answer. They provide the basis for the intensional answer that will accompany the extensional answer.

In the previous example, the two constraints may be restated as the following null database views: "employees in the design department who earn under $30,000" and "employees in researcher positions who are not in the design department".

When the employees in the design department are retrieved, we may infer from the first null view that the answer view "employees who earn under $30,000" is also null. The first component of the example intensional answer given earlier is simply a restatement of this conclusion.

Our method for inferring views of the answer that are always null may be extended to infer also views of the database that are contained entirely in the answer. When the employees in the design department are retrieved, we may infer from the second null view that the database view "employees in researcher positions" is contained in the answer in its entirety. This conclusion provided the second component of that intensional answer.

Altogether, the intensional answers characterize the extensional answers in two ways: (1) with *constraints*– views of the extensional answer that are null; and (2) with *containments*– views that are contained entirely in the extensional answer.

The remainder of this paper is organized as follows. In the next section we discuss related research works. Section 3 defines the language with which constraints are expressed, and Section 4 shows how constraints are stored. Section 5 defines the algebraic manipulations of constraints to yield constraints that are applicable to the answer. Section 6 shows how this process can be extended to infer containments. Section 7 demonstrates these method with several examples, and Section 8 concludes with a brief discussion of further refinements of the method. This representation and manipulation of views was first introduced in [6], and has been modified for the particular problem at hand.

2 Research on Intensional Answers

Recently, several research contributions have been concerned with related issues. While each adopts a different approach, all share a common goal, which is to answer queries more abstractly.

A logic database is composed of extensional predicates (*facts*) and intensional predicates (*rules*). Cholvy and Demolombe [1] are interested in providing answers to queries that are independent of a particular set of facts; i.e., answers that are derived only from the rules. The authors define a constructive derivation process that computes such answers. One of the problems they consider is how to avoid answers that are "irrelevant".

Also in the environment of logic databases, Imielinski [3] argues that rules should be allowed to occur in the answer to the query. This is shown to be beneficial both from the conceptual and the computational point of view. As an example, assume database predicates Teach(x,y), denoting that professor x can teach course y, and Group(x,z), denoting that professors x and z belong to the same research group. And assume a rule that requires that professors from the same group be able to teach the same courses. Then a query "who can teach the Database course" may be answered intensionally by "everybody in Smith's group". Exhaustive enumeration of this answer will be performed upon request.

Corella [2] notes that while research on knowledge representation produced much work on the derivation of taxonomies of concepts, concepts are also essential in responses to queries. The author defines a formal model of "semantic" retrieval (similar to intensional answers). Roughly, a concept is a unary predicate over a given domain, and a taxonomy is a finite tree of concepts, where the concept described by each node is subsumed by the concept described by its parent, and the union of sibling concepts is equal to their parent concept.

Shum and Muntz also note that answers that are exhaustive enumerations of individual objects are not always the most efficient or the most effective means of information exchange. In [7] they are concerned with implicit representation of answers through concise expressions that involve both concepts and individuals. For example, an acceptable answer to the query "Who earns more than \$30,000?" is "all engineers except John Smith". In [8] they are concerned with providing aggregate responses, where preciseness is sacrificed for conciseness. An example would be an answer such as "90/120 engineers + 20/30 managers". In both cases they assume, like Corella, the availability of taxonomies that encompass all concepts and individuals.

3 Views, Queries and Constraints

We assume the following definition of a relational database [4]. A *relation scheme* R is a finite set of *attributes* $A_1, \ldots, A_m$. With each attribute A_i a set of values D_i, called the *domain* of A_i, is associated (domains are non-empty, finite or countably infinite sets). A *relation* on the relation scheme R is a subset of the product of the domains associated with the attributes of R. A *database scheme* $\mathcal{R}$ is a set of relation schemes $R_1, \ldots, R_n$. A database instance $\mathbf{D}$ of the database scheme $\mathcal{R}$ is a set of relations $R_1(\mathbf{D}), \ldots, R_n(\mathbf{D})$, where each $R_i(\mathbf{D})$ is a relation on the relation scheme R_i.

A *view* V is an expression in the relation schemes of $\mathcal{R}$ that defines a new relation scheme, and for each database instance $\mathbf{D}$ defines a unique relation on this scheme denoted $V(\mathbf{D})$. In this paper we consider views that are defined by *conjunctive relational calculus expressions* [9]. Using domain relational calculus, expressions from this family have the form:

$$\{a_1, \ldots, a_n \mid (\exists b_1) \ldots (\exists b_k)\, \psi_1 \wedge \ldots \wedge \psi_m\}$$

Where the ψ's may be of two kinds:

1. **membership:** $(c_1, \ldots, c_p) \in R$, where R is a database relation (of arity p), and the c's are either a's or b's or constants.
2. **comparative:** $d_1\, \theta\, d_2$, where d_1 is either an a or a b, d_2 is either an a or a b or a constant, and θ is a comparator (e.g., $<, \leq, >, \geq, =, \neq$).

In particular, each a and each b must appear at least once among the c's.

We shall refer to such views as *conjunctive views*. While this family is a strict subset of the relational calculus, it is a powerful subset. The family of conjunctive relational calculus expressions is precisely the family of relational algebra expressions with the operations *product*, *selection* and *projection* (where the selection expressions are conjunctive). The attributes that participate in the selection predicate will be called *selection attributes*, and the attributes that participate in the projection will be called *projection attributes*.

In this paper, conjunctive views serve to express both queries and constraints. When used as a query, a conjunctive view defines a relation that should be derived and delivered to the user. When used as a constraint, a conjunctive view defines a derived relation that should always be null. This follows from the fact that every constraint of the form $(\forall x_1) \ldots (\forall x_n)\, (\alpha(x_1, \ldots, x_n) \Longrightarrow \beta(x_1, \ldots, x_n))$, where x_i are domain variables and α and

β are safe relational calculus expressions with these free variables, may be rewritten as a null view: $\{x_1, \ldots, x_n \mid \alpha(x_1, \ldots, x_n) \wedge \neg\beta(x_1, \ldots, x_n)\} = \emptyset$.

As an example, consider a database with the following relation schemes:

EMPLOYEE = (NAME, POSITION, SALARY, DEPARTMENT)
DEPARTMENT = (D_NAME, SUPERVISOR, DIVISION, BUDGET)

The following constraints state, respectively, that employees in the design department are guaranteed a salary of at least \$30,000, that employees in researcher positions are in the design department, that departments in the same division have the same budget, and that employees cannot earn more than their supervisors (all variables are quantified universally):

$(x_1, x_2, x_3, x_4) \in \text{EMPLOYEE} \wedge (x_4 = \text{design}) \Longrightarrow (x_3 \geq \$30,000)$

$(x_1, x_2, x_3, x_4) \in \text{EMPLOYEE} \wedge (x_2 = \text{researcher}) \Longrightarrow (x_4 = \text{design})$

$(y_1, y_2, y_3, y_4) \in \text{DEPARTMENT} \wedge (y_5, y_6, y_7, y_8) \in \text{DEPARTMENT} \wedge (y_3 = y_7) \Longrightarrow (y_4 = y_8)$

$(x_1, x_2, x_3, x_4) \in \text{EMPLOYEE} \wedge (x_5, x_6, x_7, x_8) \in \text{EMPLOYEE} \wedge (y_1, y_2, y_3, y_4) \in \text{DEPARTMENT} \wedge (x_4 = y_1) \wedge (x_5 = y_2) \Longrightarrow (x_3 \leq x_7)$

These constraints may be restated as the following null conjunctive views:

$\{x_1, \ldots, x_4 \mid (x_1, x_2, x_3, x_4) \in \text{EMPLOYEE} \wedge (x_4 = \text{design}) \wedge (x_3 < \$30,000)\}$

$\{x_1, \ldots, x_4 \mid (x_1, x_2, x_3, x_4) \in \text{EMPLOYEE} \wedge (x_2 = \text{researcher}) \wedge (x_4 \neq \text{design})\}$

$\{y_1, \ldots, y_4, y_5, \ldots, y_8 \mid (y_1, y_2, y_3, y_4) \in \text{DEPARTMENT} \wedge (y_5, y_6, y_7, y_8) \in \text{DEPARTMENT} \wedge (y_3 = y_7) \wedge (y_4 \neq y_8)\}$

$\{x_1, \ldots, x_4, x_5, \ldots, x_8, y_1, \ldots, y_4 \mid (x_1, x_2, x_3, x_4) \in \text{EMPLOYEE} \wedge (x_5, x_6, x_7, x_8) \in \text{EMPLOYEE} \wedge (y_1, y_2, y_3, y_4) \in \text{DEPARTMENT} \wedge (x_4 = y_1) \wedge (x_5 = y_2) \wedge (x_3 > x_7)\}$

For clarity, these constraints may be specified with an equivalent **disallow** statement:

```
disallow (EMPLOYEE)
where EMPLOYEE.DEPARTMENT = design
  and EMPLOYEE.SALARY < $30,000

disallow (EMPLOYEE)
where EMPLOYEE.POSITION = researcher
  and EMPLOYEE.DEPARTMENT ≠ design

disallow (DEPARTMENT:1, DEPARTMENT:2)
where DEPARTMENT:1.DIVISION =
      DEPARTMENT:2.DIVISION
  and DEPARTMENT:1.BUDGET ≠
      DEPARTMENT:2.BUDGET

disallow (EMPLOYEE:1, EMPLOYEE:2, DEPARTMENT)
where EMPLOYEE:1.DEPARTMENT =
      DEPARTMENT.D_NAME
  and EMPLOYEE:2.NAME =
      DEPARTMENT.SUPERVISOR
  and EMPLOYEE:1.SALARY > EMPLOYEE:2.SALARY
```

Thus, each **disallow** statement specifies a set of relations and a condition, disallowing tuples in these relations that satisfy the condition. The last two statements show how to handle cases where several membership subformulas reference the same relation. Note that constraints are stated on entire relations, not on particular attributes from these relations; disallowing a combination of tuples that satisfies a condition is equivalent to disallowing particular attributes from these relations that satisfy this condition. Thus, constraints do not apply projections, and are product-selection expressions.

As queries are also expressed with conjunctive views, they may be specified with a similar **retrieve** statement, as in the following query for the names and departments of all employees of the Pacific division:

```
retrieve (EMPLOYEE.NAME, EMPLOYEE.DEPARTMENT)
where EMPLOYEE.DEPARTMENT =
      DEPARTMENT.D_NAME
  and DEPARTMENT.DIVISION = pacific
```

Finally, given a view, we define every view derived from it by a selection and a projection as its *subview*. In particular, every view is its own subview.

4 Storing Constraints

Constraints are stored in new relations that are added to the database. For each database relation R a *meta-relation* R' is added. The scheme of R' is identical to the scheme of R. Also, an auxiliary relation is defined: COMPARISON = (X, COMPARE, Y). In the previous example, the database is extended with the following relations:

EMPLOYEE′ = (NAME, POSITION, SALARY, DEPARTMENT)

DEPARTMENT′ = (D_NAME, SUPERVISOR, DIVISION, BUDGET)

COMPARISON = (X, COMPARE, Y)

Relations EMPLOYEE′ and DEPARTMENT′ will be used to store membership subformulas of constraints. Comparative subformulas will be stored in relation COMPARISON.

Consider a constraint C,

$$C = \{a_1, \ldots, a_n \mid (\exists b_1) \ldots (\exists b_m)\, \psi_1 \wedge \ldots \wedge \psi_k\}$$

A subformula ψ of the kind $(c_1, \ldots, c_p) \in R$ is first modified so that the c's that are variables (i.e., a's or b's) that appear only once in the whole expression are replaced with ⊔ (blank). Hence, each component of the modified subformula is either a constant (a value), or a variable, or a blank. This *meta-tuple* is stored in the meta-relation R'. A subformula ψ of the kind $d_1\, \theta\, d_2$, where θ is not $=$, is transformed into the tuple (d_1, θ, d_2) and is stored in the auxiliary relation COMPARISON. If θ is $=$, then all occurrences of d_1 in the other subformulas are substituted with d_2.

This representation of views in relations recalls the representation of queries in QBE [10]. As an example, Figure 1 shows an instance of the example database extended with the previous four constraints. For convenience of presentation, each pair of relations R, R' is shown as a single contiguous table.

Note that each individual meta-tuple may be regarded as defining a subview of the corresponding relation. The constants and variables specify the selection condition (and the entire relation is projected). For example, the meta-tuple (⊔, ⊔, x_1, design) stored in EMPLOYEE′ specifies a selection of all tuples of relation EMPLOYEE for which DEPARTMENT = design and SALARY = x_1. (A variable shared by another meta-tuple, such as x_1, specifies a selection condition which is satisfied by any value from a set of values defined elsewhere.)

EMPLOYEE

NAME	POSITION	SALARY	DEPARTMENT
Brown	engineer	35,000	manufacture
Green	technician	28,000	service
Jones	manager	51,000	design
King	manager	45,000	sales
Scott	manager	48,000	service
Smith	researcher	40,000	design
Wilson	manager	50,000	manufacture
Wood	salesperson	26,000	sales
		x_1	design
	researcher		x_2
		x_6	x_7
x_8		x_9	

DEPARTMENT

D_NAME	SUPERVISOR	DIVISION	BUDGET
design	Jones	pacific	275
manufacture	Wilson	pacific	275
sales	King	atlantic	160
service	Scott	atlantic	160
		x_3	x_4
		x_3	x_5
x_7	x_8		

COMPARISON

X	COMPARE	Y
x_1	$<$	30,000
x_2	$\neq$	design
x_4	$\neq$	x_5
x_6	$>$	x_9

Figure 1: Database Extended with Constraints

5 Manipulating Constraints

Assume a database $R_1, \ldots, R_n$ and a meta-database $R'_1, \ldots, R'_n$, COMPARISON. Let $C_1, \ldots, C_m$ be constraints (null views) defined on this database.

Let Q be a conjunctive query against this database. We are interested in constraints that are defined on its answer. That is, we are interested in subviews of Q that are null.

We describe a method that discovers such subviews. Basically, this method *generates* views of $C_1, \ldots, C_m$ that are subviews of Q, by manipulating the definitions of $C_1, \ldots, C_m$ algebraically. These manipulations mirror those that are necessary to implement Q. In effect, we *generalize* the standard product, selection and projection operations to manipulate also relations of view definitions. Clearly, every view of null views is also null, so the subviews of Q that are generated in this way are indeed null.

This method is illustrated by the commutative diagram shown in Figure 2. The solid lines describe the current situation: the null views R' define constraints on the database relations R, and the virtual relation A is derived from R to answer query Q. The dashed lines describe our method: query processing is extended to manipulate also R' to yield the null views A' that define constraints on the answer A.

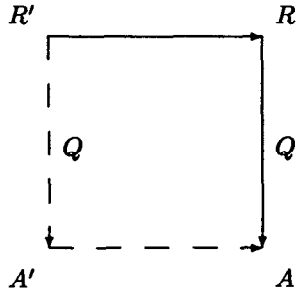

Figure 2: Extending Query Processing to Constraints

5.1 Meta-Relation Operations

Definition 1: Assume that R' and S' are meta-relations that define, respectively, views of R and S. The product of R' and S', denoted $R' \times S'$, is defined as follows. For every pair r and s of meta-tuples from R' and S', respectively,

$$\begin{aligned} r &= (a_1, \ldots, a_m) \\ s &= (b_1, \ldots, b_n) \end{aligned}$$

$R' \times S'$ includes the meta-tuple:

$$q = (a_1, \ldots, a_m, b_1, \ldots, b_n)$$

Proposition 1: Let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$, $s(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by r, s and q. Then $q(\mathbf{D}) = r(\mathbf{D}) \times s(\mathbf{D})$.
Proof: Let λ and μ denote, respectively, the selection predicates of r and s. Then $r(\mathbf{D})$, $s(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following product-selection expressions:

$$\begin{aligned} r(\mathbf{D}) &= \sigma_\lambda(R(\mathbf{D})) \\ s(\mathbf{D}) &= \sigma_\mu(S(\mathbf{D})) \\ q(\mathbf{D}) &= \sigma_{\lambda \wedge \mu}(R(\mathbf{D}) \times S(\mathbf{D})) \end{aligned}$$

We observe that $\sigma_{\lambda\wedge\mu}(R(\mathbf{D}) \times S(\mathbf{D})) = \sigma_\lambda(R(\mathbf{D})) \times \sigma_\mu(S(\mathbf{D}))$.

Definition 2: Assume that R' is a meta-relation that defines views of R. Let λ denote a primitive selection predicate (i.e., either $A'_i \,\theta\, c$, or $A'_i \,\theta\, A'_j$). The selection from R' by predicate λ, denoted $\sigma_\lambda(R')$, is defined as follows. Consider first the case $\lambda = A_i \,\theta\, c$, and let r be a meta-tuple from R',

$$r = (a_1, \ldots, a_i, \ldots, a_m)$$

Denote by μ the selection predicate expressed by a_i [1]. $\sigma_\lambda(R')$ includes the meta-tuple:

$$q = (a_1, \ldots, a'_i, \ldots, a_m)$$

where a_i represents $\lambda \wedge \mu$. Consider now the case $\lambda = A_i \,\theta\, A_j$, and let r be a meta-tuple from R',

$$r = (a_1, \ldots, a_i, \ldots, a_j, \ldots, a_m)$$

Denote by μ the selection predicate expressed by a_i and a_j. $\sigma_\lambda(R')$ includes the meta-tuple:

$$q = (a_1, \ldots, a'_i, \ldots, a'_j, \ldots, a_m)$$

where a'_i and a'_j represent $\lambda \wedge \mu$.

Proposition 2: Let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by r and q. Then $q(\mathbf{D}) = \sigma_\lambda r(\mathbf{D})$.
Proof: $r(\mathbf{D})$ and $q(\mathbf{D})$ can be expressed as the following selection expressions:

$$\begin{aligned} r(\mathbf{D}) &= \sigma_\mu(R(\mathbf{D})) \\ q(\mathbf{D}) &= \sigma_{\mu\wedge\lambda}(R(\mathbf{D})) \end{aligned}$$

We observe that $\sigma_{\mu\wedge\lambda}(R(\mathbf{D})) = \sigma_\lambda\sigma_\mu(R(\mathbf{D}))$.

Definition 3: Assume that R' is a meta-relation that defines views of R. The projection of R' that removes its i'th attribute, denoted $\pi_{R-A_i}(R')$, is defined as follows. For every meta-tuple r from R',

$$r = (a_1, \ldots, a_m)$$

If a_i is $\sqcup$, then $\pi_{R-A_i}(R')$ includes the meta-tuple:

$$q = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m)$$

Proposition 3: Let $\mathbf{D}$ be an instance of this database, and let $r(\mathbf{D})$ and $q(\mathbf{D})$ denote, respectively, the relations defined by the meta-tuples r and q. Then $q(\mathbf{D}) = \pi_{R-A_i}(r(\mathbf{D}))$ [2].
Proof: Let λ denote the selection predicate of r. $r(\mathbf{D})$

[1] If a_i is blank, then μ is true.

[2] In general, we define $\pi_\alpha(R)$ as a projection on those attributes in α that are in R. Thus, if attribute A_i had already been removed, a projection on $R - A_i$ has no effect.

and $q(\mathbf{D})$ can be expressed as the following selection-projection expressions:

$$r(\mathbf{D}) = \sigma_\lambda(R(\mathbf{D}))$$
$$q(\mathbf{D}) = \sigma_\lambda \pi_{R-Ai}(R(\mathbf{D}))$$

We observe that if the i'th attribute of R does not participate in the predicate λ, then $\sigma_\lambda \pi_{R-Ai}(R(\mathbf{D})) = \pi_{R-Ai}\pi_\alpha\sigma_\lambda(R(\mathbf{D}))$.

Note that projection retains only meta-tuples whose projected attributes are disjoint from their selection attributes.

Propositions 1, 2 and 3 are summarized in the following theorem:

Theorem: Assume a database $R_1, \ldots, R_n$ and a meta-database $R'_1, \ldots, R'_n$, COMPARISON, with constraints (null views) $C_1, \ldots, C_m$. Let Q be a conjunctive query against this database. Let S be the relational algebra expression that implements Q. Let S' be the relational algebra expression obtained from S by substituting every reference to R with a reference to R'. S operates on the relations to yield the answer A. S' operates on the meta-relations to yield the meta-answer A'. Then, the meta-tuples in A' define subviews of A that are also views of $C_1, \ldots, C_m$.

The theorem guarantees that meta-tuples in A' define subviews of A that are views of $C_1, \ldots, C_m$ (and are therefore null). However, some meta-tuples may still contain references to meta-tuples outside A', and are therefore not expressible entirely within A'. Such subviews are avoided if S' is modified so that all products are performed first, and their result is pruned to retain only those meta-tuples that do not contain references to other meta-tuples. Also, as we explain below, it is advantageous to perform selections before projections. Altogether, S' is transformed to a sequence of products, followed by selections, and ending with projections. This simple strategy for implementing conjunctive queries is not necessarily optimal. However, we note that the optimality is not so essential for meta-relations, because they are relatively small. For the actual relations, where optimality is essential, a different strategy may be implemented.

5.2 Refinements

The theorem guarantees that the method for generating subviews is *sound*, but it does not guarantee that it is *complete*. That is, this method generates subviews of the result that are indeed null, but does not necessarily generate all such subviews. A method that would guarantee completeness would undoubtedly be of a different complexity altogether. Yet, with several simple refinements, it can be improved to generate additional desirable subviews. Two such refinements are sketched below. Both address the "loss of views" that occurs during projection, when meta-tuples are discarded if they restrict (with variables or constants) the attributes that are removed.

The first refinement modifies the product operation. Assume that Q is a product of R and S, followed by a projection that removes all the attributes of S. Obviously, Q is equivalent to R, and A' should retain all the meta-tuples of R'. However, these meta-tuples may be discarded by the projection, if they contain restrictions in the attributes contributed by S'. To handle this situation we may extend the product of meta-relations to include also these two tuples:

$$q_1 = (a_1, \ldots, a_m, \sqcup, \ldots, \sqcup)$$
$$q_2 = (\sqcup, \ldots, \sqcup, b_1, \ldots, b_n)$$

These tuples define all previous subviews of R and S as subviews of the product of R and S.

The second refinement modifies the selection operation. As defined, this operation requires conjuncting μ, the predicate expressed in the meta-tuple, with λ, the predicate expressed in the query. However, as all the tuples in the resulting relation satisfy λ, the expression $\mu \wedge \lambda$ is simply μ. Therefore, it appears that a simpler definition of the selection operation may be provided, which simply retains all meta-tuples without any modification. On the other hand, this simpler definition often would not generate the best definitions of views, nor would it detect views that are indeed irrelevant. As an example, assume a meta-tuple that selects the employees whose salaries are between \$30,000 and \$60,000, and consider the following four queries that select the employees whose salaries are (1) between \$20,000 and \$40,000, (2) between \$20,000 and \$70,000, (3) between \$40,000 and \$50,000, and (4) under \$30,000. In each case, the given view (employees whose salaries are between \$30,000 and \$60,000) could be retained as a view of the employees selected. However, it would be more desirable to handle this selection on a case by case basis, as follows. In the first query, modify the given view to define the employees whose salaries are between \$30,000 and \$40,000; in the second query, retain the given view without any modification; in the third query, modify the given view so it does not restrict the salary at all; and, in the fourth query, discard the given view altogether. In general, we observe four different cases: If λ *implies* μ, the meta-tuple is selected and the corresponding field is cleared (i.e., the variable or the constant is replaced

by $\sqcup$); if μ *implies* λ, the meta-tuple is selected without any modification; if λ and μ are *contradictory*, the meta-tuple is discarded; otherwise, the meta-tuple is selected, and is modified to represent $\mu \wedge \lambda$. Clearing selection predicates ensures that more meta-tuples will "survive" future projections. Determining the appropriate case for given μ and λ may require consulting relation COMPARISON, and, possibly, modifying it. While for most views and queries this task is quite simple, if an implementation chooses not to determine the case for predicates of certain form, then in those cases the relevant meta-tuple must not be selected. Note that the only other time where relation COMPARISON is used, is when the views in A' are described to the user.

6 Detecting Containments

Consider again the selection operation, and let $(a_1, \ldots, a_n)$ be a meta-tuple in the result of the product. This meta-tuple represents a *denial* of a conjunction of n primitive predicates. Denote by μ_i the predicate represented by a_i [3]. Then this meta-tuple represents the constraint:

$$\neg(\mu_1 \wedge \cdots \wedge \mu_n)$$

which may be formulated as an implication:

$$\mu_1 \wedge \cdots \wedge \mu_{j-1} \wedge \mu_{j+1} \wedge \cdots \wedge \mu_n \Longrightarrow \neg\mu_j$$

Let $\lambda_1 \wedge \cdots \wedge \lambda_m$ be the selection predicate expressed in the query. As discussed earlier, selection is performed with a *sequence* of steps, each comparing a μ_i with a λ_j. μ_i and λ_j are related in one of four different ways:

1. $\lambda_j \Longrightarrow \mu_i$
2. $\mu_j \Longrightarrow \lambda_i$
3. $\neg(\lambda_j \wedge \mu_i)$
4. Otherwise

In all but case 3, λ_j and μ_i are not contradictory, and the meta-tuple is retained (possibly modified). A meta-tuple that "survives" the entire selection sequence describes a constraint on the result.

Consider now case 3. It may be formulated as $\lambda_j \Longrightarrow \neg\mu_i$. Obviously, when this selection step terminates, the tuples in the result satisfy λ_j. Hence, they also satisfy $\neg\mu_i$. Recall that the meta-tuple may be represented as the implication:

$$\mu_1 \wedge \cdots \wedge \mu_{i-1} \wedge \mu_{i+1} \wedge \cdots \wedge \mu_n \Longrightarrow \neg\mu_i$$

[3] If $a_i = \sqcup$, then $\mu_i = true$.

Thus, the tuples that satisfy $\mu_1 \wedge \cdots \wedge \mu_{i-1} \wedge \mu_{i+1} \wedge \cdots \wedge \mu_n$ must satisfy $\neg\mu_i$. Since the tuples of the result indeed satisfy $\neg\mu_i$, all the tuples that satisfy $\mu_1 \wedge \cdots \wedge \mu_{i-1} \wedge \mu_{i+1} \wedge \cdots \wedge \mu_n$ are included in the result. Therefore, in case 3, the conjunction of all the constraint predicates except μ_i represents a view that is contained in the result.

We now extend the previous selection process. Consider a selection step that compares μ_i with λ_j. Case 1, 2 and 4 are treated as before. In case 3, the meta-tuple is not discarded; instead, a_i (which represents the predicate μ_i) is replaced by the symbol $\neg$, to indicate that it is never satisfied in the result. Future comparisons should leave this value unchanged. When an attribute that is removed by the projection contains $\neg$, the meta-tuple is retained, but this $\neg$ is "attached" to the meta-tuple.

7 Examples

When the entire processing ends, each meta-tuple in the result is examined. If it does not include any $\neg$, it is a constraint. If it includes exactly one $\neg$, it is a containment.

Constraints are translated into **disallow** statements; containments are translated into **contain** statements. The syntax of these statements is similar: the words **disallow** or **contain** are followed by an expression in the attributes of the result, similar to the **where** clause of a **retrieve** statement. This set of statements constitutes the *intensional* answer, and it accompanies the usual *extensional* answer.

Example 1: Consider the query of the introduction, to retrieve the names, positions and salaries of employees in the design department:

retrieve (EMPLOYEE.NAME, EMPLOYEE.POSITION, EMPLOYEE.SALARY)
where EMPLOYEE.DEPARTMENT = **design**

This query may be implemented with the following sequence of algebraic operations:

1. $\text{A} \leftarrow \sigma_{(\text{DEPARTMENT=design})}(\text{EMPLOYEE})$
2. $\text{A} \leftarrow \pi_{\text{NAME,POSITION,SALARY}}(\text{A})$

The same operations that are applied to the database relations are applied to their meta-relations counterparts:

1. $\text{A}' \leftarrow \sigma_{(\text{DEPARTMENT=design})}(\text{EMPLOYEE}')$
2. $\text{A}' \leftarrow \pi_{\text{NAME,POSITION,SALARY}}(\text{A}')$

This sequence does not require products. Before se-

lection, the meta-relation EMPLOYEE′ is pruned to include only those meta-tuples that do not reference other meta-tuples:

EMPLOYEE′

NAME	POSITION	SALARY	DEPARTMENT
		x_1	design
	researcher		x_2

The selection involves a single query predicate DEPARTMENT=**design**. In the first meta-tuple the constraint predicate is also DEPARTMENT=**design**. This corresponds to case 1, and the DEPARTMENT field is cleared. In the second meta-tuple the constraint predicate is DEPARTMENT≠**design**. This corresponds to case 3, and the DEPARTMENT field is replaced by ¬. We have:

A′

NAME	POSITION	SALARY	DEPARTMENT
		x_1	
	researcher		¬

Finally, the projection retains both meta-tuples:

A′

	NAME	POSITION	SALARY
			x_1
¬		researcher	

In response to a request to retrieve the names, positions and salaries of employees in the design department, the system issues this extensional answer:

NAME	POSITION	SALARY
Jones	manager	51,000
Smith	researcher	40,000

And this intensional answer:

disallow SALARY < $30,000
contain POSITION = researcher

These statements inform the user that all employees retrieved earn at least $30,000, and that all employees in researcher positions were retrieved.

Example 2: Consider a query to retrieve pairs of different departments that are in the same division, and their budgets:

retrieve (DEPARTMENT:1.D_NAME,
DEPARTMENT:1.BUDGET,
DEPARTMENT:2.D_NAME,
DEPARTMENT:2.BUDGET)
where DEPARTMENT:1.DIVISION =
DEPARTMENT:2.DIVISION
and DEPARTMENT:1.D_NAME <
DEPARTMENT:2.D_NAME

This query may be implemented with the following sequence of algebraic operations [4]:

1. A ← DEPARTMENT × DEPARTMENT
2. A ← $\sigma_{(\text{DIV:1=DIV:2})\wedge(\text{D\_NAME:1<D\_NAME:2})}(\text{A})$
3. A ← $\pi_{\text{D\_NAME:1,BUD:1,D\_NAME:2,BUD:2}}(\text{A})$

The same operations that are applied to the database relations are applied to their meta-relations counterparts:

1. A′ ← DEPARTMENT′ × DEPARTMENT′
2. A′ ← $\sigma_{(\text{DIV:1=DIV:2})\wedge(\text{D\_NAME:1>D\_NAME:2})}(\text{A}')$
3. A′ ← $\pi_{\text{D\_NAME:1,BUD:1,D\_NAME:2,BUD:2}}(\text{A}')$

The result of the product after outside references are removed is:

A′

D_N:1	S:1	D:1	B:1	D_N:2	S:2	D:2	B:2
		x_3	x_4			x_3	x_5
		x_3	x_5			x_3	x_4

The selection involves two query predicates. The first predicate is DIVISION:1=DIVISION:2. In both meta-tuples the constraint predicate is also DIVISION:1=DIVISION:2. This corresponds to case 1, and in each meta-tuple the DIVISION fields are cleared. The second predicate is D_NAME:1<D_NAME:2. In both meta-tuples the constraint predicate is *true*. Again, this corresponds to case 1, and in each meta-tuple the D_NAME fields remain clear. We have:

A′

D_N:1	S:1	D:1	B:1	D_N:2	S:2	D:2	B:2
			x_4				x_5
			x_5				x_4

Finally, the projection retains both meta-tuples:

A′

D_NAME:1	BUDGET:1	D_NAME:2	BUDGET:2
	x_4		x_5
	x_4		x_4

In response to a request to retrieve pairs of different departments that are in the same division, the system issues this extensional answer:

D_NAME:1	BUDGET:1	D_NAME:2	BUDGET:2
design	275	manufacture	275
sales	160	service	160

And this intensional answer:

disallow BUDGET:1 ≠ BUDGET:2

This statement (generated twice) informs the user that the budgets must be equal.

[4] When a relation has several attributes named A, then A : i denotes the i'th appearance of A.

8 Conclusion

We presented a method that addresses the important issue of providing more meaningful answers to queries. The relational model, which stores data in relations, is extended to store constraints in meta-relations, and the algebra of relations is extended to an algebra of meta-relations. Each query is processed against both the database and the meta-database, yielding an answer and a meta-answer. The meta-answer describes constraints and containments that apply to the answer.

Currently, the methods handle only conjunctive views (constraints, containments, and queries). Work is underway on extensions that will handle more general views; for example, views with *disjunctions* and views with *aggregate functions*. Since the methods do not necessarily detect all the constraints and containments that apply to the result, the intensional answers that are provided should be considered sound characterizations, that are not necessarily complete.

One of the difficult problems in providing intensional answers is how to identify the statements that are *relevant* to a query. For example, consider a query to list all employees and their salaries. While it is true that the set of employees listed satisfies the constraint that all those in researcher positions are in the design department, this information is probably irrelevant. It is possible to extend the model to include in the database additional knowledge that will assist in determining the relevant statements. A satisfactory implementation of this approach requires additional investigation.

An alternative solution is to define an intensional statement as relevant, if it can be expressed entirely with the attributes retrieved by the query. Thus, the constraint that all employees in researcher positions are in the design department is relevant only to queries that retrieve both POSITION and DEPARTMENT. While this solution may not be entirely satisfactory, it provides an extremely simple pruning mechanism, which is usually effective. Indeed, the projection operation implements this very pruning strategy, as it discards the meta-tuples whose projection attributes and selection attributes intersect, retaining only constraints and containments that are expressed entirely with the attributes retrieved by the query.

Another problem that remains to be solved is how to prune intensional answers, so they do not include statements that are implied by other statements. For example, assume a constraint that all female employees earn over \$30,000, and a constraint that all employees over 35 years old earn over \$40,000, and consider a query to list the female employees over 35 years old. The meta-answer will include two constraints: all employees listed earn over \$30,000, and all employees listed earn over \$40,000. Obviously, the former statement is redundant.

Work is also underway on a database "front-end" interface that will implement our methods and enable experimentation. The user will define constraints with **disallow** statements, and the system will insert automatically the appropriate meta-tuples into the meta-relations. In response to a **retrieve** statement, the user will receive the usual extensional answer, accompanied by inferred **disallow** and **contain** statements. Thus, the meta-relations and meta-tuple notation would be completely transparent, with all user-system communication done with customary query language statements.

Acknowledgement. The author is grateful to Alex Borgida for pointing out several relevant works.

References

[1] L. Cholvy and R. Demolombe. Querying a rule base. In *Proc. First Int. Conf. on Expert Database Systems*, April 1986, pages 365–371.

[2] F. Corella. Semantic retrieval and levels of abstraction. In *Proc. First Int. Workshop on Expert Database Systems*, October 1984, pages 91–114.

[3] T. Imielinski. Intelligent query answering in rule based systems. *Journal of Logic Programming*, 4(3):229–257, September 1987.

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

[5] A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *Proc. Fifth Int. Conf. on Data Engineering*, February 1989, pages 339–347.

[6] A. Motro. Integrity = validity + completeness. (To appear in ACM Trans. on Database Systems.)

[7] C. D. Shum and R. Muntz. Implicit representation for extensional answers. In *Proc. Second Int. Conf. on Expert Database Systems*, April 1988, pages 257–273.

[8] C. D. Shum and R. Muntz. An information-theoretic study on aggregate responses. In *Proc. Fourteenth Int. Conf. on Very Large Data Bases*, August 1988, pages 479–490.

[9] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, 1982.

[10] M. Zloof. Query-by-Example: a database language. *IBM Systems Journal*, 16(4):324–343, December 1977.