

# The Semantics of Answering Queries Using Views, Information Integration, CSCI 6967-01

## 1 Integrity Constraints

Assume we are given the local as view model over a global schema. So far, we have disregarded the properties of the global model, such as keys and integrity constraints. Recall an integrity constraint of the form:

$$\{A_1, \dots, A_n\} \rightarrow \{B_1, \dots, B_m\}$$

where  $A_1, \dots, A_n$  and  $B_1, \dots, B_m$  are attributes in the global schema is read as: whenever two tuples have the same values for attributes for  $A_1, \dots, A_n$ , then they will have the same values for attributes  $B_1, \dots, B_m$ . A simple case of an integrity constraint is a key constraint. If  $A_1, \dots, A_n$  is a key, then it should imply all the attributes in a relation.

Foreign key constraints are another example of integrity constraints that say that the values in a specific set of attributes for a relation must come from (the primary key of) another relation. We represent foreign keys as:

$$R1(A_1, \dots, A_n) \subseteq R2(B_1, \dots, B_n)$$

In this case,  $A_1, \dots, A_n$  is a foreign key in relation  $R1$  that references the primary key  $B_1, \dots, B_n$  of relation  $R2$ .

### 1.1 Need for recursive rewritings

Suppose we are given the following model:

*schedule(Airline, FlightNo, Date, Pilot, Aircraft)*

and the following functional dependencies:

*Pilot*  $\rightarrow$  *Airline*

*Aircraft*  $\rightarrow$  *Airline*

which means that a pilot only flies in a single airline and each aircraft belongs to a single airline. Suppose now we are given the following view:

$v(D, P, C) : \neg \text{schedule}(A, N, D, P, C)$

which records which pilot flies on which date and which aircraft. Suppose now we consider a query asking for pilot that work for the same airline as Mike.

$$q(P) : \text{--schedule}(A, N, D, 'mike', C), \text{schedule}(A, N', D', P, C')$$

Note that we cannot check which airline Mike works for, but we know that if Mike and Ann fly on the same aircraft, then they must work for the same airline.

$$q(P) : \text{--v}(D, 'mike', C), v(D_1, P, C)$$

We also know that any aircraft Ann flies in is also for the same airline as 'mike'. So, we can find all pilots who flies in an airplane as Ann. Suppose John is such a pilot.

$$q(P_2) : \text{--v}(D, 'mike', C), v(D_1, P, C), v(D_2, P, C_2), v(D_3, P_2, C_2)$$

Now, we can extend this further by finding all pilots that flew in an airplane John flew in.

$$q(P_3) : \text{--v}(D, 'mike', C), v(D_1, P, C), v(D_2, P, C_2), v(D_3, P_2, C_2), v(D_4, P_2, C_3), v(D_5, P_3, C_3)$$

It is easy to see that to find all the possible answers to this query, we need to apply this method recursively (possibly infinite amount of times).

To accomplish this, we will rewrite the original view using Skolem functions:

$$\text{schedule}(f_1(D, P, C), f_2(D, P, C), D, P, C) : \text{--v}(D, P, C)$$

We will also introduce a rule for each functional dependency:

$$e(A, A') : \text{--schedule}(A, N, D, P, C), \text{schedule}(A', N', D', P', C'), e(P, P')$$

$$e(A, A') : \text{--schedule}(A, N, D, P, C), \text{schedule}(A', N', D', P', C'), e(C, C')$$

$$e(X, Z) : \text{--e}(X, Y), e(Y, Z)$$

$$e(X, Y) : \text{--e}(Y, Z)$$

Now, we will write query such that if  $c$  is a constant in one of the subgoals, then replace it with variable  $Z$ , and add  $e(Z, c)$  to the query body. If  $X$  is a variable in the head of the query, then replace it with  $X'$  and add  $e(X, X')$  to the query. If  $Y$  is a variable in the body of the query that does not appear in the head, but appears in at two subgoals of the query, then replace one of the  $Y$  with  $Y'$  and add  $e(Y, Y')$  to the query. Continue until we cannot apply this rule anymore.

Given these rules, we can rewrite the query as follows:

$$q(P) : \text{--schedule}(A, N, D, P^*, C), \text{schedule}(A', N', D', P', C'), e(P, P'), e(A, A'), e(P^*, mike)$$

Now, we can apply the rules computed previously to find all possible rewritings of this query:

$$\text{schedule}(f_1(D, P, C), f_2(D, P, C), D, P, C) : \text{--} \quad v(D, P, C)$$

$$e(A, A') : \text{--} \quad \text{schedule}(A, N, D, P, C), \text{schedule}(A', N', D', P', C'), e(P, P')$$

$$e(A, A') : \text{--} \quad \text{schedule}(A, N, D, P, C), \text{schedule}(A', N', D', P', C'), e(C, C')$$

$$e(X, Z) : \text{--} \quad e(X, Y), e(Y, Z)$$

$$e(X, Y) : \text{--} \quad e(Y, X)$$

$$e(X, X).$$

## 1.2 Query rewriting in the presence of integrity constraints

Suppose now we have the following data model (primary keys underlined):

*student*(*Scode*, *Sname*, *Scity*)

*university*(*Ucode*, *Uname*)

*enrolled*(*Scode*, *Ucode*)

and the following foreign key constraints:

$enrolled(Scode) \subseteq student(Scode)$

$enrolled(Ucode) \subseteq university(Ucode)$

Suppose now the information integration system contains the following view definitions:

$s_1(X, Y, Z) \leftarrow st(X, Y, Z)$

$s_2(X, Y) \leftarrow un(X, Y)$

$s_3(X, Y) \leftarrow en(X, W)$

Now, suppose the three sources are not complete with respect to the given query. We are given the following instances:

$s_1$

12	anne	florence	21
15	bill	oslo	24

$s_2$

AF	bocconi
BN	ucla

$s_3$

12	AF
16	BN

Consider the query:

$q(X) : \neg student(X, Y, Z), enrolled(X, W)$

We will get 12 using the above sets. However, if we consider  $s_1$  incomplete, then we know that there is a student with id 16 due to the integrity constraint, but we do not have it in the student relation. So, the correct answer should be 12,16.

How can we get complete information?

Expand each relation in the data model to reflect the functional dependencies, and define new version  $p'$  based on each relation  $p$  by using the integrity constraints.

$$\begin{aligned}
 student'(X, Y, Z) &: \neg student(X, Y, Z) \\
 university'(X, Y) &: \neg university(X, Y) \\
 enrolled'(X, Y) &: \neg enrolled(X, Y) \\
 student'(X, f_1(X), f_2(X)) &: \neg enrolled(X, Y) \\
 university'(X, f_3(X)) &: \neg enrolled(X, Y)
 \end{aligned}$$

Now, we can take the original query, and find all possible ways to expand it using the above rules. Rewrite the query as:

$q(X) : \neg student'(X, Y, Z), enrolled'(X, W)$

For  $student'$ , we have:

$q(X) : \neg student(X, Y, Z), enrolled'(X, W)$   
 $q(X) : \neg enrolled'(X, Y), enrolled'(X, W)$

for  $enrolled'$ , we have:

$q(X) : \neg student(X, Y, Z), enrolled(X, W)$   
 $q(X) : \neg enrolled(X, Y), enrolled(X, W)$

Now, we can rewrite each query using the sources and compute the result. The first rule above will give us 12 as before. However, the second rule will produce 12, 16. The union of both 12, 16 is the maximal answer.

Note that we will rewrite the query using these rules as long as the head of query does not contain any skolem functions.

### 1.3 Completeness descriptions

In the previous section, we used the integrity constraints to find the maximal answers to a given query if the individual sources are not guaranteed to contain all the tuples for their described content. In some cases, we may not want to execute all possible ways to execute a query when some of the queries are very costly. Instead, we can model when a source is considered to be complete. To do this, we will use constraints that relate two relations with the same scheme to each other. These constraints can involve built-in predicates over the attributes of a relation or constraints over other relations.

For example:

$LC(Movie', Movie, Year \geq 1965)$

states that  $Movie'$  is complete with respect to  $Movie$  for movies after 1965. Note that  $Movie'$  may be a source supplying the movie relation. Similarly, the expression:

$LC(Movie', Movie, Show(Title, Theater, Hour))$

states that  $Movie'$  is complete with respect to  $Movie$  for movies that appear in  $Show$  (assuming here that  $Title$  is a key for  $Movie$  and a foreign key for  $Show$ ).

We can now say that a rewriting of a query (using the quoted version of the relations) is complete if the query is guaranteed to return the same results as the query against the actual data model. The queries may also be instant complete, i.e. even if in theory a relation is not complete, it may be complete for a specific query. For example, suppose we ask the query:

$q(X) : \neg p(X, Y), r(Y, Z), Z < 10$

and  $p$  is known to be incomplete. However, suppose the following is true:

$LC(r', r, Z < 20)$

Now,  $r'$  is complete with respect to the above query. The question is then whether  $p(X, Y)$  is complete with respect to all the  $Y$  values in  $r(Y, Z), Z < 10$ . If so, then we are guaranteed that the results to the above query will be complete. We illustrate how to check each below.

To check whether the answers to a query are complete with respect to given a completeness description

$LC(R', R, C)$ , we need to check whether a tuple inserted to relation  $R$  that satisfies **NOT**  $C$  can be an answer to the query. In other words,  $Q, \mathbf{NOT} C$  should be unsatisfiable. To see why this is the case, note that **NOT**  $C$  is the part of the relation  $R$  that is not complete. So, we need to see whether we query any tuples in this part. So,  $Q$  should not return any tuples from this part. This is only possible if  $Q, \mathbf{NOT} C$  is satisfiable.

that violates the completeness condition can be an answer to the query. We simply check for all completeness conditions with constraint  $C$ , whether **NOT**  $C$  and the query together are unsatisfiable, or inconsistent.

For example, given the query:

$$q(T, D) : -movie(T, D, Y), show(T, Th, H)$$

and the completeness description:

$$LC(Movie', Movie, Year \geq 1965)$$

we check whether

$$q(T, D) : -movie(T, D, Y), show(T, Th, H), \mathbf{NOT}(Year \geq 1965)$$

is satisfiable, i.e.

$$q(T, D) : -movie(T, D, Y), show(T, Th, H), (Year < 1965)$$

since it is, then this query is not independent of the completeness specification and hence the results are not guaranteed to be complete.

To check this, we can simply rewrite each source  $S$  with information content  $D$  and completeness constraint  $C$  such that:

$$S' : -D$$

$$S' : -E, \mathbf{NOT} C$$

where  $E$  is an uninterpreted symbol. Now, checking the equivalence of programs is reduced to checking whether the query rewritten using  $S'$  instead of  $S$  is equivalent to the original query.

Checking equivalence is undecidable for recursive queries, but algorithms for non-recursive cases exist. For a special subcase of the queries where constraints only contains built-ins with one variable and constraint, then checking equivalence is possible.

To check this, we can simply find whether to solve  $Q$ , we need to prove  $E$ . This is easy to do if  $C$  contains only EDB predicates and built-in functions that have comparisons between a variable and a constant. In this case, we can check in polynomial time whether the constraints involving the built-in predicates are satisfiable.

Answer completeness for an instance checks whether a query is guaranteed to be complete for a specific query. To check for this, we must consider the functional dependencies in the data model. Given a query  $Q : H : -B, C$  where  $C$  is the constraints involving built-in predicates, we find the minimal set of variables  $\overline{X}$  that functionally determine all the others in the query based on the given functional dependencies. We now find the maximal set of predicates  $\overline{p}$  that contain all the variables in  $\overline{X}$  (if no such predicate exists, then we will not have an answer to this problem). We also find the projection  $C_1$  of the constraint  $C$  on variables in  $\overline{X}$ .

Now, we check whether the query  $Q_1 : H'(\overline{X}') : -\overline{p}, \overline{C}$  is complete with respect to the given completeness descriptions (here  $\overline{X}'$  is the set of variables that exist in  $\overline{p}$ .)

If so, now we must check whether the remainder of the query is complete with respect to  $Q_1$ . To do so, we must find the set of variables  $Y$  that appear in the remaining predicates  $\overline{p}_2$  of the query. Find the set  $\overline{Y}$  that appear in both  $Q_1$  and  $Y$ , and find the minimal set  $\overline{Y}'$  that imply all variables in  $\overline{Y}$ . Suppose  $C_2$  is the projection of  $C$  over variables in  $\overline{Y}'$ . Now, check whether:

$$H(\overline{Y}') : -\overline{p}, C_1$$

returns a superset of tuples from

$$H(\overline{Y}') : -\overline{p}_2, C_2.$$

If this is the case, then the remaining query is complete.

For example, let's us look at the query:

$$ans(Scode, Uname) : -student(Scode, Sname, Scity), enrolled(Scode, Ucode), university(Ucode, Uname), Ucode = AF', Sname \leq B$$

Now, variables  $Scode, Ucode$  functionally determine the rest. So, we write down query  $Q_1$  as:

$$Q_0 : ans(Scode, Uname) : -enrolled(Scode, Ucode), Ucode = AF'$$

and check whether this query is complete with respect to the given completeness specifications. If this is the case, now we check the remaining query.  $Scode, Uname$  still imply all the remaining variables. So, we write the following queries:

$$Q_1 : ans(Scode, Uname) : -enrolled(Scode, Ucode), Ucode = AF'$$

$$Q_2 : ans(Scode, Uname) : -student(Scode, Sname, Scity), university(Ucode, Uname), Ucode = AF'$$

Now, we check whether  $Q_2$  contains everything in  $Q_1$ . Note that  $Q_1$  returns *anne, bocconi*, whereas  $Q_2$  returns *anne, bocconi* and *bill, bocconi*. As  $Q_1 \subseteq Q_2$ , we say that the above query is complete if  $Q_0$  above is complete.

## 1.4 Tackling inconsistencies

Suppose we are given a set of integrity constraints, but the results returned by the queries violate these constraints. For example, suppose we are given:

$$CourseRoom(Ccode, SectionCode, RoomCode, TimeCode), SeminarRoom(SCode, RoomCode, TimeCode)$$

$$CourseRoom(X, S1, Y1, T), CourseRoom(X, S2, Y2, T) \rightarrow Y1 = Y2$$

$$CourseRoom(X, S, Y, T1), CourseRoom(X, S, Y, T2) \rightarrow T1 = T2$$

$$CourseRoom(X1, S1, Y1, T), CourseRoom(X2, S2, Y1, T) \rightarrow Y1 \neq Y2$$

$$CourseRoom(X1, S1, Y1, T), SeminarRoom(X2, Y2, T) \rightarrow Y1 \neq Y2$$

which tells that two sections of a course that meet at the same time should be scheduled in the same room and a course and seminar that meets at the same cannot be scheduled at the same room.

Now, suppose we are given the following source descriptions:

$$s1(C1, S1, T) : -CourseRoom(C1, S1, a, T)$$

$$s2(C1, S1, X) : -CourseRoom(C1, S1, X, t1)$$

$s3(C1, S1, T) : \neg SeminarRoom(C1, S1, T)$

Suppose, the following are the contents of each source:

$s_1$	<table border="1" style="border-collapse: collapse;"> <tr><td>abc</td><td>s1</td><td>t1</td></tr> <tr><td>def</td><td>s1</td><td>t2</td></tr> </table>	abc	s1	t1	def	s1	t2
abc	s1	t1					
def	s1	t2					

$s_2$	<table border="1" style="border-collapse: collapse;"> <tr><td>abc</td><td>s2</td><td>b</td></tr> <tr><td>def</td><td>s1</td><td>a</td></tr> </table>	abc	s2	b	def	s1	a
abc	s2	b					
def	s1	a					

$s_3$	<table border="1" style="border-collapse: collapse;"> <tr><td>sem1</td><td>b</td><td>t1</td></tr> <tr><td>sem2</td><td>a</td><td>t2</td></tr> </table>	sem1	b	t1	sem2	a	t2
sem1	b	t1					
sem2	a	t2					

Now, given the above specifications, we have basically the following facts we can derive from the sources:

- $A : CourseRoom(abc, s1, a, t1)$
- $B : CourseRoom(abc, s2, b, t1)$
- $C : CourseRoom(def, s1, a, t2)$
- $D : CourseRoom(def, s1, a, t1)$
- $E : SeminarRoom(sem1, b, t1)$
- $F : SeminarRoom(sem2, a, t2)$

which has many inconsistencies. How do we resolve these? The answer is to throw away facts that conflict with the given query but only if it is necessary. In other words, find the largest set of consistent facts. In other words, we find the largest set of facts such that we cannot add any more without causing inconsistency. Note that there can be more than one such maximal set.

In the above example, for example, facts  $\{A, C, E\}$  form such maximal set. Similarly,  $\{B, D, F\}$ ,  $\{A, E, F\}$  and  $\{D, E, F\}$  are also maximal sets. Given these are maximal and inherently consistent, we can now develop methods to decide whether a fact is true by either picking one of these models or finding facts that exist in all (or most) models. The best interpretation is dependent on the application.

Suppose now we trust the correctness of each source to a different degree. To model this, we are going to assume an ordering among the sources,  $s_1 > s_2$  means that  $s_1$  is preferable to  $s_2$ . Given an ordering among all the sources, we now define an ordering among two sets of facts  $W_1, W_2$  as follows:

Suppose  $W_1 = W \cup a \cup b$  and  $W_2 = W \cup c$  where  $a, b, c$  are sets of facts where each  $a, b, c$  can be empty. Then, we say that  $W_1 \preceq W_2$  iff there is a one-to-one and onto mapping  $h$  from  $a$  to  $c$  such that if a fact  $x \in a$  is from source  $s_x$  and  $h(x) \in c$  is from source  $s_y$  then  $s_x \geq s_y$  in the given ordering of resources.

In other words  $W_1 \preceq W_2$  if  $W_2$  contains more facts or replaces each fact in  $W_1$  with a fact that comes from a more reliable source.

In the above example, suppose we are given that  $s_3 > s_2 > s_1$ . Then, we can say  $\{A, C, E\} \preceq \{B, D, F\}$ , since  $A$  is from  $s_1$  and is replaced by  $B$  from source  $s_2$ , similar relationship is true for  $C, D$ .  $E, F$  are equally preferable. Similarly, we have:

- $\{A, C, E\} \preceq \{A, E, F\}$
- $\{B, D, F\} \preceq \{D, E, F\}$
- $\{A, C, E\} \preceq \{D, E, F\}$
- $\{A, E, F\} \preceq \{D, E, F\}$

Given these we can conclude that  $\{D, E, F\}$  is the maximal set given these preferences and hence can be used to define the semantics of the system.

Computing such a maximal set is potentially exponential in general. Heuristic algorithms are needed to find approximate answers.

## 2 Modeling source capabilities

Note that the LAV approach generally models the information contained in a source. It does not describe how these results can be obtained from the sources. The sources may have different capabilities. For example, given a view of the form:

$$V(X, Y, Z) : -p(X, Y, Z)$$

we are saying that the source returns  $X, Y, Z$  attributes. However, if the source is just a flat file, then we can find all possible combinations of  $X, Y, Z$  attributes. However, if the source is a query service, it may require the user to input a specific attribute and then return the remaining attributes as a result. For example, IMDB allows the user to enter the title of a movie and then it will return all the other properties of the movie relation. We represent such a relation as:

$$v_1^{bff}(X, Y, Z) : -p(X, Y, Z)$$

$$v_2^{fb}(W, T) : -r(W, T)$$

where  $b$  means that a variable must be bound to a constant and  $f$  means that the variable does not have to be bound, i.e. it is free. This is called a binding pattern for this source. For a single source, there may be multiple ways to query it and hence, we may have binding patterns.

Given source descriptions with binding patterns, we can now find a rewriting of the query where each source predicate (such as  $v$  above) is adorned with a binding pattern.

Suppose we are given:

$$q^{bf}(X, Y, W) : -p(X, Y, Z), r(W, Z)$$

which means that the query will supply a binding for  $X$  as user input. In this case, we can rewrite the query as follows:

$$q^{bf}(X, Y, W) : -v_1^{bff}(X, Y, Z), v_2^{fb}(W, Z)$$

We can say that a rewriting is valid if (1) the predicates in the body can be ordered in such a way that the first predicate has a valid binding pattern with respect to the query head, and each following predicate has a valid binding pattern with respect to the previous predicate (where the outputs of these predicates can be assumed to be bound in the following predicates), and (2) the bound values of all binding patterns in the rewriting agree with the bound values for the source descriptions.

Finding a valid rewriting is NP-complete for conjunctive queries and source descriptions without built-in predicates. However, if a rewriting is given, finding if it has a valid rewriting for a given set of source descriptions is polynomial.

Note that this model does not take into account that some sources may allow the user not to only select for equality, but also perform inequality searches. For example, given a rewriting:

$$q(X, Y, Z) : -v_1(X, Y), Z > 10$$

if the source allows the user to search on  $Z > 10$  even if it does not return  $Z$ , then we can still use the above rewriting to find the answers to this query. To model this type of capability, Information manifold uses a different model. For each source, it lists which variables (or combination of these) the source can do equality or selection (such as  $>$ ) search on, the minimum number of inputs that must be provided and the maximum number of inputs that can be provided. Note that this model lacks the precision of the above model, but the interpretation is similar. Now, given a rewriting, we first check whether any

selection conditions can be pushed to the source. This may be crucial for making rewritings feasible, but at the same time, it may be useful for reducing the amount of information returned by the sources (hence it is a query optimization method). The aim is to push as many selection and equality conditions to the sources as possible. Hence, we must find an ordering of predicates that is feasible and reduces the amount of information that must be retrieved from the sources.