Completeness Information and Its Application to Query Processing

Amihai Motro

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

Abstract

The assumption that a database includes a representation of every occurrence in the real world environment that it models (the *Closed World Assumption*) is frequently unrealistic, because it is always made on the database *as a whole*. This paper introduces a new type of database information, called ***completeness information***, to describe the ***subsets*** of the database for which this assumption is correct. With completeness information it is possible to determine whether each answer to a user query is complete, or whether any subsets of it are complete. To users, answers which are accompanied by a statement about their completeness are more meaningful. First, the principles of completeness information are defined formally, using an abstract data model. Then, specific methods are described for implementing completeness information in the relational model. With these methods, each relational algebra query can be accompanied with an instantaneous verdict on its completeness (or on the completeness of some of its subsets).

1. Introduction

The question whether a database contains all the occurrences of data which it attempts to model has received much attention lately. The assumption that it does, called the *Closed World Assumption (CWA)*, allows one to conclude that when a query cannot be satisfied in the database, then the answer to it is negative [3]. The CWA is always made on the database *as a whole*. In practice, however, this assumption is not realistic. Most databases include at least some information which is possibly incomplete. In other words, in reality the CWA can only be made on some ***subsets*** of the database.

In this paper we introduce a new kind of database information, which we call ***completeness information***. With completeness information a database system can determine whether each answer to a user query is complete (i.e. all the real world occurrences are represented), or whether any ***subsets*** of it are complete (i.e. all the real world occurrences which satisfy some additional constraints are represented). As an example, consider a database on recordings of music, and assume it is known to include complete information on all recordings by domestic record companies. The answer to a query to retrieve all recordings of Beethoven symphonies on the CBS label, can be shown to be complete. The answer to a query to retrieve all recordings which feature Toscanini, can be shown to be complete with regard to domestic recordings only. Clearly, answers which are accompanied by statements about their completeness are more meaningful.

Completeness information and its application to query processing are first defined and discussed using an abstract model of databases. We give formal definitions for such notions as *completeness assertions*, *complete answers* and *partially complete answers*. However, to develop specific methods for expressing, storing, manipulating and applying completeness information, a particular data model must be adopted, and we focus on the relational model. A major concern is that the application of completeness information to query processing should be efficient, avoiding, for example, methods that require mechanical theorem proving. Towards this end, we develop a simple method for storing completeness information in the relations themselves, and show how to extend relational algebra operations to *preserve* this information. Thus, the determination of the completeness of answers is available as an inexpensive by-product of standard query processing.

It is interesting to compare this new concept of completeness information with the way databases handle *incomplete information*. Often, some of the information about the real world that a database attempts to model is unavailable. Consider, for example, a relational database with relation R: $A_1,\ldots,A_N$. Each tuple $(a_1,\ldots,a_n)$ of R models a particular real world relationship. However, it may be that a certain real world relationship is known to exist, but not all the a values are available. Such missing values are usually modeled by *null values*. Much research has been done on the different types of null values, and on the manipulation of databases with null values (for a review of this topic see [1]).

In many respects null values are very much like *data* (the database operations are extended to handle this new type of data satisfactorily). On the other hand, the description of the information which is complete is actually *meta-data*[1]. Another observation is that incomplete information and completeness information are somewhat complementary: an item of incomplete information states that there exists data which could not be specified as "regular data"; an item of completeness information states that there does not exist any data which has not been specified already.

[1]It may be interesting to note here, that our methods for specifying and storing completeness information for relational databases will make it *appear* like data.

The next section gives formal definitions of completeness information and its application to query processing. Section 3 and 4 discuss specific techniques for relational databases, and Section 5 concludes with a brief summary and discussion of some further issues.

2. Completeness Information

Assume a relational database on music with two relations:

```
RECORDING = (LABEL, NUMBER, COMPOSER,
             COMPOSITION, CONDUCTOR)
COMPOSER =  (NAME, NATIONALITY,
             YEAR-OF-BIRTH, YEAR-OF-DEATH)
```

RECORDING describes all recordings of music currently available, and COMPOSER contains data on composers of music. These relations are models of real world recordings and composers. However, there is a significant difference between these two models: while it is possible that RECORDING covers the complete catalogue of available recordings, we cannot expect COMPOSER to include all composers that have ever lived (e.g. many composers may be unknown outside their own small circles). This suggests that database information may be classified into two kinds: information which is *complete*, and information which is only *partial*.

In the above example, it was quite simple to describe which information was complete, and which was not: the database knew about every recording, but not about every composer. In general, however, the description of the complete information may be more complex. In that example, the database could include complete information only on recordings on the CBS label, or on recordings of compositions by French composers. In general, to express completeness of information requires a language whose power is equivalent to the power of a query language: in the same way that a query describes the set of data items that satisfy a certain requirement, a completeness assertion would declare that the database contains all the data items that satisfy a certain requirement.

Clearly, the meaning of completeness depends on the semantics of the database. For example, if the relation COMPOSER was meant to model, not all composers, but only, say, those composers whose compositions will be performed this season by the Los Angeles Philharmonic, then it may well be complete. We assume that the semantics of the database are known when the completeness assertions are defined.

In this section we present formal definitions for completeness information and related aspects. Our formalizations use the notion of *query generalization*, as defined in [2].

Let $\mathcal{F}$ denote the universe of all possible facts. A *specification* s is a characteristic function on $\mathcal{F}$:

$$s : \mathcal{F} \rightarrow \{0,1\}.$$

Given a set of facts $F \subseteq \mathcal{F}$, we define the facts of F specified by s as:

$$s(F) = s^{-1}(1) \cap F.$$

A specification s on $\mathcal{F}$ is a *completeness assertion* for a set of facts $F \subseteq \mathcal{F}$, if $s^{-1}(1) \subseteq F$. The subset $s^{-1}(1)$ is then said to be *complete* in F.

A *database* D is a pair $<F,A>$, where $F \subseteq \mathcal{F}$ is a set of facts and A is a set of completeness assertions. Each completeness assertion $a \in A$ defines a subset of F which is complete.

Specifications are also used to define retrieval requests. A *query* q against database $D = <F,A>$ is a specification on $\mathcal{F}$. The set $q(F)$ is called the *answer* to q.

Given two specifications s_1 and s_2, we define s_2 to be *more general* than s_1 (and s_1 to be *more specific* than s_2), denoted $s_1 \prec s_2$, if $s_1(1) \subseteq s_2(1)$.

For example, consider these five specifications:

1. All French composers born after 1900.
2. All French composers born after 1850.
3. All composers born after 1900.
4. All composers.
5. All persons.

In this set of specifications, 2 and 3 are more general than 1, 4 is more general than either 2 or 3, and 5 is more general than 4. These specifications demonstrate some methods to generalize specifications: weaken a condition (from 1 to 2), remove a condition altogether (from 2 to 3, or from 3 to 4), and substitute a more general concept (from 4 to 5).

Clearly, when a query fails in a given database (i.e. evaluates to the empty set), then every more specific query would fail, but more general queries may succeed; and when a query succeeds, then every more general query would succeed, but more specific queries may fail. When a completeness assertion holds in a given database, then every more specific assertion also holds. Consequently, completeness information need only include the most general assertions.

Given two specifications s_1 and s_2, we define the *intersection specification* $s_1 \cap s_2$ and the *union specification* $s_1 \cup s_2$ in the standard way: for any fact $f \in \mathcal{F}$, $s_1 \cap s_2(f)=1$ if both $s_1(f)=1$ and $s_2(f)=1$, and 0 otherwise; $s_1 \cup s_2(f)=1$ if either $s_1(f)=1$ or $s_2(f)=1$, and 0 otherwise.

For example, consider these five specifications:

1. All French composers.
2. All composers who wrote operas.
3. All composers who wrote incidental music.

4. All French composers who wrote operas.

5. All composers who wrote operas or incidental music.

In this set of specifications, 4 is the intersection of 1 and 2, and 5 is the union of 2 and 3.

Clearly, an intersection specification is more specific than either of the two participating specifications. Consequently, when a completeness assertion holds in a database, then its intersection with any other specification also holds.

Consider now a query q against database $\mathcal{D} = <F,A>$. As before, the query can simply be evaluated against F, producing the answer $q(F)$. However, we can also determine the completeness of this answer, by comparing it with the completeness assertions A. Formally, the answer to q is *complete* if there exist assertions $a_1,...,a_n$ in A, such that $q \prec \cup_{i=1}^{n} a_i$; otherwise, the answer is *partial*.

In the above example, assume the database includes the following completeness assertions:

- All recordings on the CBS label.
- All recordings of compositions by French composers.

then queries such as

- All recordings of Beethoven's 9th Symphony on CBS.
- All recordings of compositions by 19th Century French Composers.

can be shown to have complete answers.

Consider now a query q which is not generalized by any union of completeness assertions. If q intersects with a completeness assertion, then the information described by the intersection specification is complete. Formally, the answer to q is *partially complete with regard to* $a \in A$ if $q \cap a \neq \emptyset$ (i.e. there exists $f \in \mathcal{F}$, such that $q \cap a(f)=1$).

In the above example, assume the database includes the following completeness assertions:

- All recordings of compositions by French composers.
- All recordings of compositions by German composers.

And consider the following query:

- All recordings of operas.

While the answer to this query is not complete, it is partially complete with regard to operas that were composed by French or German composers.

Completeness information improves the responsiveness of the system, as an answer which is accompanied with a statement about on its completeness (or partial completeness) is more meaningful. Completeness information is particularly useful within a mechanism for handling failed queries, such as the one described in [2]. When a user query fails to match any data, but the answer can be shown to be *complete*, then the user can be notified authoritatively that data to satisfy his query does not exist (and therefore the premise for the query is incorrect).

3. Completeness Information in Relational Databases

In this and the following section we focus on relational databases, and how completeness information can be expressed, stored, manipulated and applied in this particular model. We note that at several places we sacrifice generality, for simplicity, efficiency and economy.

In the relational model, the concepts of facts and database subsets are selected to be, respectively, tuples and relations. Note that subsets cannot be arbitrary collections of facts, as relations can only have tuples of the same dimension. We define one database subset (i.e. relation) to be contained in another, if the former can be obtained from the latter with a selection and a projection.

Domain relational calculus [4] offers a convenient tool for expressing specifications (data queries or completeness assertions). Each specification is an expression of the form $\{ x_1,\ldots,x_n \mid \psi(x_1,\ldots,x_n) \}$, where $x_1,\ldots,x_n$ are domain variables and ψ is a safe formula in predicate logic with $x_1,\ldots,x_n$ as its only free variables. It defines a database subset, which is the n-tuples of data items that satisfy ψ. Thus, one may think of the completeness assertions as defining another database, which contains only complete information. By the previous definition of containment, one specification is a generalization of another, if the relation it defines can be reduced with a selection and a projection to the relation defined by the other specification.

Consider now a completeness assertion $A = \{ x_1,\ldots,x_n \mid \psi_1(x_1,\ldots,x_n) \}$ and a query $Q = \{ y_1,\ldots,y_m \mid \psi_2(y_1,\ldots,y_m) \}$. To determine whether A is more general than Q, one needs to prove that under any interpretation $\psi_2 \Rightarrow \psi_1$. Attempting this with mechanical theorem proving is a possibility. However, this solution is quite impractical, as it requires substantial computation and is undecidable.

We describe here an altogether different approach, which avoids such problems. First, we limit the complexity of completeness assertions to assertions of the following kind:

$$\{ a_1,\ldots,a_n \mid (\exists b_1)\ldots(\exists b_m)\ \mathrm{R}_1(c_{11},\ldots,c_{1k_1}) \wedge \ldots \wedge \mathrm{R}_p(c_{p1},\ldots,c_{pk_p}) \}$$

where the a's and b's are variables, each appearing at least once among the c's, and the c's, which are not a's or b's, are data items (constants). Such an assertion states the completeness of a particular subset of the database which is derived from relations $\mathrm{R}_1,\ldots,\mathrm{R}_p$ (note that the relations R_i are not necessarily distinct).

For example, in the previous database, the following expression asserts the completeness of the set of all numbers of recordings on the CBS label:

$$\{ a \mid (\exists b_1)(\exists b_2)(\exists b_3)\ \mathrm{RECORDING}(\mathbf{CBS},a,b_1,b_2,b_3) \}.$$

And to assert the completeness of all recordings (label and number) of compositions by French composers:

$$\{ a_1,a_2 \mid (\exists b_1)(\exists b_2)(\exists b_3)(\exists b_4)(\exists b_5)\ \mathrm{RECORDING}(a_1,a_2,b_1,b_2,b_3) \wedge \mathrm{COMPOSER}(b_1,\mathbf{French},b_4,b_5) \}.$$

Next, we describe a method for storing completeness assertions from this family *in the relations themselves.* As we shall point out later, this approach provides important advantages, over separate storage of this assertions as logic expressions. Our method recalls the representation of QBE queries in skeleton tables [5] .

A completeness assertion of the general type defined above is represented as p ***information tuples***; each tuple $(c_{i1},\ldots,c_{ik_i})$ is stored in relation R_i. The c's which are a's are suffixed with a *; the c's which are variables (a's or b's) that appear only once in this assertion are replaced with blanks. Thus, each component of an information tuple may be either a constant (a data item), or a variable, or a blank, or a variable suffixed by a *, or a *. A constant or a variable impose restrictions on the values that attribute may have. A blank is an attribute whose value is irrelevant to this assertion. A * (whether appended to a variable or to a blank) is an attribute on which completeness is asserted.

We can assume that when an assertion involves more than one information tuple, each tuple will share a variable with at least one other tuple (otherwise, the assertion can be regarded as several independent assertions). Therefore, if all variables used in assertions are chosen to be different, then the information tuples belonging to each assertion can be identified and recovered from the relations. A relation with information tuples will be called an *extended* relation.

As an example, Figure 3-1 shows the two assertions stated above together with a small instance of our database (which clearly does not satisfy the completeness claimed by the assertions).

RECORDING

LABEL	NUMBER	COMPOSER	COMPOSITION	CONDUCTOR
Angel	S-3613	Bizet	Carmen	Beecham
CBS	M2S-794	Beethoven	Symphony_No_9	Bernstein
CBS	M-31838	Tchaikovsky	Swan_Lake	Ormandy
DGG	2531250	Beethoven	Violin_Concerto	Karajan
London	LDR-71090	Saint-Saens	Organ_Symphony	Dutoit
Philips	9500359	Debussy	La_Mer	Haitink
RCA	VICS-6113E	Verdi	Aida	Toscanini
RCA	LSC-6199	Bizet	Carmen	Karajan
CBS	*			
*	*	x		

COMPOSER

NAME	NATIONALITY	YEAR-OF-BIRTH	YEAR-OF-DEATH
Beethoven	German	1770	1827
Bizet	French	1838	1875
Debussy	French	1862	1918
Mozart	Austrian	1756	1791
Saint-Saens	French	1835	1921
Tchaikovsky	Russian	1840	1893
Verdi	Italian	1813	1901
x	French		

Figure 3-1: A Database Extended with Completeness Information

This method for storing completeness information has several advantages. Firstly, the specification of completeness assertions using QBE-like notation is very intuitive. Secondly, storing the information does not require any new data structures. Thirdly, the information may be updated with the same tools used to update the data. But the greatest advantage is in the testing of completeness. As our general definitions indicated, data and completeness information are separate concepts, and query processing requires evaluating each query against both. The advantage of storing both in the same structures, is that we can develop methods with which completeness determination becomes a "by product" of the standard query evaluation.

Consider the extended relations in the above example, and assume that we apply relational algebra operations to these relations. A question that has to be addressed is how these operations should handle the information tuples stored in these relations. Of course, we would like these operations to manipulate the extended relation so that the information tuples in the resulting relation would describe its completeness correctly.

Let Q be a database query involving database relations I, and let I' be the resulting relation. Let A be the assertions that describe the completeness of I. We are interested in applying Q also to A, producing information A' that will describe correctly the completeness of I'. This goal is sketched in the following diagram, where solid lines show existing conditions and dashed lines show how the diagram should be completed:

$$\begin{tikzcd} A \arrow[r] \arrow[d, dashed, "Q"] & I \arrow[d, "Q"] \\ A' \arrow[r, dashed] & I' \end{tikzcd}$$

Achieving this goal would guarantee that after a query had been processed, the verdict on the completeness of the answer is immediately available in the result. For example, if the result contains an information tuple which has * in every column, then the answer is complete; if it contains a tuple with the data item **a** in attribute A and * in every other column, then the answer is partially complete with regard to A=**a**; and so on. The extension of relational algebra operators to *preserve* completeness information is described in the next section.

4. Extending Relational Algebra to Preserve Completeness Information

For brevity, we describe here how to extend three relational algebra operators: selection, projection and Cartesian product (and therefore join). Other operators should present no special difficulty.

We noted that the meaning of each completeness assertion depends on the semantics of the relations that are involved. Consequently, the semantics of the database relations must be determined before completeness information is defined. To show that completeness information is preserved after each extended operation, it is important to determine the semantics of the output relation, in terms of the semantics of the input relation(s).

Consider, for example, a relation R, which models a particular real world concept. After a selection σ_F is applied to R, the output relation should no longer be considered a model of the original concept, but of a more restricted one. Thus, if COMPOSER models all composers, then $\sigma_{\text{NATIONALITY} = \textbf{German}}$(COMPOSER) models only the German composers. Consequently, although both relations may have the same attributes, an information tuple in the output relation asserts something different from the same information tuple in the input relation.

4.1. Cartesian Product

Our extension of the Cartesian product is to handle information tuples as if they were data tuples. We observe that the Cartesian product of two relation subsets which are complete, is a complete subset in the output relation. In particular, the Cartesian product of a subset which is complete and a single data tuple, is also complete.

Let T be the Cartesian product of two extended relations R and S. T includes information tuples of two kinds: tuples that were obtained from an information tuple and a data tuple, and tuples that were obtained from two information tuples. Let α be an information tuple from R, describing the complete subset R_α, let β be a data tuple from S, and let γ be their product tuple (i.e. concatenation). Each tuple of R_α appears in T suffixed by the tuple β. This subset of T is also complete, and is described by the information tuple γ. Assume now that β is not a data tuple but an information tuple, describing the complete subset S_β. Their Cartesian product is a complete subset of T, and is described by the information tuple γ.

4.2. Projection

In discussing projection, we consider only projections which remove a single attribute (the treatment of general projections is quite similar). To extend projection to information tuples, we distinguish between information tuples that *restrict* the values of the attribute which is being removed (i.e. either a constant, or a variable) and those that do not (i.e. a blank). In the former case we retain the (projected) information tuple; in the latter case we discard it.

Let R_α be a complete subset of R, described by the information tuple α. Let R', R'_α, and α' denote the corresponding structures after attribute A had been removed. While R'_α, is a complete subset of R', it is not necessarily described by α'. If α had a ***restriction*** on attribute A, then α', having removed the restriction, now describes a subset which is possibly *larger* than R'_α, and not necessarily complete. We can be certain that α' actually describes R'_α, only if α had no restrictions on attribute A.

4.3. Selection

In discussing selection we assume that the selection formula is a single equality comparator (the generalization to more complex formulas is straightforward). Again, to extend selection to information tuples, we distinguish between information tuples that *restrict* the values of the selection attribute (i.e. either a constant, or a variable) and those that do not (i.e. a blank). In the former case the information tuple will be selected only if its value matches the value to which it is being compared; in the latter case the tuple will always be selected, and the value to which it is being compared will be substituted in the tuple. We observe that after a selection, the output relation may either preserve a complete subset, or it may reduce it; but in either case the new subset is complete, with regard to the semantics of the output relation.

Let R_α be a complete subset of relation R, described by the information tuple α. Consider now the operation $\sigma_{A=\mathbf{a}}(R)$. The tuples of R_α that satisfy $A=\mathbf{a}$ will be selected and, as mentioned above, will form a complete subset of the output relation. α itself will be selected if in attribute A it has either **a** or blank, but in both cases will have $A=\mathbf{a}$ in the output relation. In the former case, the tuples of R_α will all have $A=\mathbf{a}$, and therefore R_α will be preserved, along with α, in the output relation. In the latter case, only the tuples of R_α for which $A=\mathbf{a}$ will be selected, and these will be described correctly by the information tuple obtained from α by substituting **a** in attribute A. Similarly, if the selection formula compares two attributes, e.g. $\sigma_{A=B}(R)$, then α will be selected, unless both attributes are restricted by constants which are different. Except when both attributes are restricted by the same constant, α will be modified: if both attributes are unrestricted, then the same new variable will be substituted in both attributes; if only one is unrestricted, then the restricting value (variable or constant) will be substituted in the unrestricted attribute; if both are restricted by variables, then one of the variables will be substituted by the other throughout; and if one is a variable and the other a constant, then the variable will be substituted by the constant throughout.

We now demonstrate our methods with two examples. Consider this query to list the recordings (label and number) of compositions by Beethoven:

$$\pi_{\text{LABEL,NUMBER}}(\sigma_{\text{COMPOSER} = \textbf{Beethoven}}(\text{RECORDING}))$$

The extended operations produce the following relation:

LABEL	NUMBER
CBS	M2S-794
DGG	2531250
CBS	*

The information tuple indicates that this answer is complete only in regard to recordings on the CBS label.

As another example, consider this query to list the recordings (label, number and conductor) of compositions of 19th century French composers:

$$\pi_{\text{LABEL,NUMBER,CONDUCTOR}}(\sigma_{(\text{NATIONALITY} = \textbf{French})\wedge(\textbf{1800} \leq \text{YEAR-OF-BIRTH} \leq \textbf{1850})}((\text{RECORDING} \times \text{COMPOSER})))$$

Even though this query uses a natural join and a complex selection predicate, one can easily to verify that its answer is:

LABEL	NUMBER	CONDUCTOR
Angel	S-3613	Beecham
London	LDR-71090	Dutoit
RCA	LSC-6199	Karajan
*	*	

The information tuple indicates that the answer is complete in the first two attributes (i.e. the projection on the first two columns is complete). However, if we assume that the first two attributes form a key, then the whole answer is in fact complete.

5. Conclusion

We have presented a new kind of database information, for specifying the subsets of the database which are "closed world". We have laid a formal basis, with definitions of *specification* (completeness assertion or query), *intersection* and *union* of specifications, *generalization* relationship between specifications, *completeness* of answers, and *partial completeness* with regard to an assertion.

We have then focused on completeness information in relational databases, and defined a simple family of domain calculus expressions, for expressing a large number of popular completeness assertions. For these assertions we have defined a representation (similar to QBE) that enabled us to store the information along with the data. We have then shown how to extend relational algebra to manipulate these extended relations, so that the information tuples in the output relations assert correctly the completeness of the result. The main advantage of this representation, besides it simplicity and economy, is that the standard query evaluation process generates, as a by-product, the verdict on the completeness or partial completeness of the answer.

There are several other issues that require further investigation. One is the relationship between incomplete information and completeness information (for example, how does one interpret completeness assertions in the presence of tuples with null values). Another is the relationship between functional dependencies and completeness information (for example, we note that a completeness assertion on the key attributes of a relation implies that complete information is available on the other attributes, as well). A third set of issues involves our implementation of completeness information in the relational model, which was motivated by simplicity, efficiency and economy, but at the cost of generality. An interesting issue here is how much "deduction power" was sacrificed in this implementation, in comparison with an implementation that would express completeness assertions with general first order logic formulas, and use mechanical theorem proving to determine whether a user query has a complete answer. There are also pragmatic issues, such as the update of data which is covered by completeness assertions, and the update of the completeness assertions themselves.

Acknowledgements

I am grateful to my colleague Rick Hull for his important comments.

References

[1] D. Maier.
The Theory of Relational Databases.
Computer Science Press, 1983, Chapter 12.

[2] A. Motro.
Query Generalization: A Technique for Handling Query Failure.
In *Proceedings of the First International Workshop on Expert Database Systems*, pages 314-325. Kiawah Island, South Carolina, 1984.

[3] R. Reiter.
Towards a Logical Reconstruction of Relational Database Theory.
In M. L. Brodie, J. Mylopoulos and J. W. Schmidt (editors), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, chapter 8. Springer-Verlag, 1984.

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

[5] M. Zloof.
Query-by-Example: A Database Language.
IBM Systems Journal 16(4):324-343, December, 1977.