RESEARCH PAPER | Computer Science | Volume : 2 | Issue : 3 | December 2012 | ISSN - 2249-555X

Uncertainty Lineage Databases - A case Study

KEYWORDS

Mr. Vishal K. Pandya	Dr. Dhaval R. Kathiriya
(HOD, Shri V J Modha College Of IT)	Director, Information Technology Anand Agricultural University, Anand

ABSTRACT *Uncertainty Lineage Database (ULDB), Lineage enables simple and consistent representation of uncertain data, it correlates uncertainty in query results with uncertainty in the input data and query processing with lineage and uncertainty together presents computational benefits over treating them individual. Lineage identifies a data item's derivation, in terms of other data in the database or outside data sources. In this case study we combine lineage and uncertainty into one data model.*

Fundamental
Lineage is also important for uncertainty within a single database. One relationship between uncertainty and lineage is that lineage can be used for understanding and resolving uncertainty. To draw a loose analogy with web search, correctness of answers returned by a search engine is uncertain, reflected by their ranking.

Search engine basically provide lineage information including at least a URL and text snippet, and users tend to consider both ranking and lineage to determine which links to follow. Many applications that integrate information from multiple sources may be uncertain about which data is correct and the original source and derivation of data may offer helpful additional information. Lineage is also important for uncertainty within a single database, when any users pose queries against uncertain data, the results are uncertain too. Lineage facilitates the correlation and coordination of uncertainty in query results with uncertainty in the input data. For instance we know that either one set of base data is correct or another one is but not both. Then we do not want to produce any query results that are derived by combine data from the two sets, directly or indirectly. Lineage is a particularly convenient and intuitive mechanism for encoding the complex uncertainty relationship that can arise among base and derived data.

In this we formalize query processing on ULDBs at currently algorithms for running a wide class of relational operators and describe how these algorithms can be join to execute complex / difficult de

Databases with Lineage
Here describing databases with lineage, which we can say LDBs. LDBs and ULDBs extend the relational model, each relation is a multiset of tuples. We continue to attach unique identifiers to each tuple in the database. A set of relations R = {R1,…..,Rn} in a database, user can use I(R) to denote all identifiers in relations R1,…..Rn.

The lineage of a tuple identifies the data from which it was derived. Some tuples are LDB are derived from other LDB tuples, like output of queries. The lineage of derived tuples consists of references to other tuples in the LDB, through their unique identifiers, base tuple in some type of cases may be derived from entities outside the LDB, like external data set or a sensor feed. For the next case we bring in external lineage. External Lineage refers to a set of external symbols we denote by E. The set of symbols known to an LDB is S=I(R) E.

For example, we introduce as a running example from Examination Copy-Solver database. Consider LDB relations Student(Stud_Name, Paper) and Observer(Ob_Name, Paper) representing student information and subject (paper) sightings respectively. Consider also a relation Check (Ob_Name, Stud_Name) produced by the query Ob_Name,Stud_Name(Observer [X] Student). Here is some sample data like:

Observer: Student:

ID	Ob Name	Paper	ID	Stud_Name	Paper
01	BKP	C	11	VKP	Java
02	SBP	C++	12	SDB	C
03	BKP	Java	13	HBM	Java
			14	VKP	C++

Check:

ID	Ob_Name	Stud_Name	Paper
21	BKP	SDB	C
22	BKP	VKP	Java
23	BKP	HBM	Java
24	SBP VKP		C++

Operations:

$\lambda(21) = 01 \wedge 12$	$\lambda(22) = 03 \wedge 11$
$\lambda(23) = 03 \wedge 13$	$\lambda(24) = 02 \wedge 14$

However operations are performed there is often an lineage function for the tuples in the result. The above example present a natural lineage function for joins: Lineage of a tuple t in the result of a join is a conjunction of the set of tuples from each of the joined relations, that were combined to form t, e.g. BKP, SBP, BBM is obtained from BKP,C and SDB,C. Some operation such as select, project, join and union, create only conjunctive lineage. However some other operations create more complex lineage formulas. For instance the difference operator creates negations in the Boolean formula, while duplicate elimination creates disjunctions. In an LDB query results lineage that refers to other tuples in the database. In our model t he result of applying a query Q to database D includes the original relations R and new relation for Q's answer with the appropriate lineage function. While a relation may contain duplicates, each tuple has its own lineage. For example , tuples 22 and 23 in above example have same data but each one has different derivation and therefore different lineage. It we perform a duplicate elimination on Check we obtain one 22 23.

We specify algorithms for all relational operators, in our case study the emphasis is on readability particularly for understanding how lineage is generated. Implementations as real physical operators would instead focus on efficiency. We specify our operators so that each one unfolds lineage as it produces results. When the relations S1,….,Sn at the leaves of the plan are base relations we treat all of their tuples to have λ (t)= t. Thus the unfolded lineage that is output by every operator. Including the root, refers to data in the input relations. That is we never generate lineage referring to intermediate results within a single query. If we have a derived input relation S, we require query result lineage to refer to

Volume : 2 | Issue : 3 | December 2012 | ISSN - 2249-555X

tuples in S, rather than further unfolding S's lineage. When executing a query plan we treat the lineage of all input tuples t as λ (t)=t.

Selection and Projection
Selection: Consider σ operator applied to relation S with lineage λS. We produce the result R with lineage λ as under given,

(1) For each tuple t in S with at least one alternative satisfying the selection predicate, add a tuple to R containing all alternatives in t satisfying t he selection predicate,
(2) For each alternative in R corresponding to alternative S in S , set λ ()= λS (S).

Projection: Consider Operator π applied to relation S with lineage λS. We produce the result R with lineage λ as under given,

(1) For each tuple in S add corresponding tuple to R with each alternative projected onto the specified attribute list.
(2) For each alternative in R corresponding to alternative αs S in S , set λ (α)= λS (S).

Join, Union And Inetersection
Join: Consider operator [X] applied to S1 and S2 in S2 if at with lineage λ1 and λ2 respectively, we produce the result R with lineage λ as under given,

(1) For each pair of tuples S1 and S2 if at least one pair of alternatives matches the join condition:

a. Construct a tuple t in R with an alternative for every pair of alternatives 1 and 2 satisfying the join condition.
b. For each alternative in t , set λ (α)= λ1 (α1) λ2 (α2).

Union: Consider operator applied to S1 and S2 with lineage λ1 and λ2, we produce the result R with lineage λ as under given.

(1) For each tuple in S1 , add an identical tuple to R.
(2) For each alternative in R corresponding to alternative α1 in S1 set λ (α)= λ1 (α1 (=λ2 (α2).

Intersection: Consider operator applied to S1 and S2 with lineage λ1 and λ2, we produce the result R with lineage λ as under given

(1) For each tuple S1:
A. Create a tuple t in R containing all alternatives S1 that appear in S2, drop t if no alternative S1 appears in S2.
b. For each alternative in t.

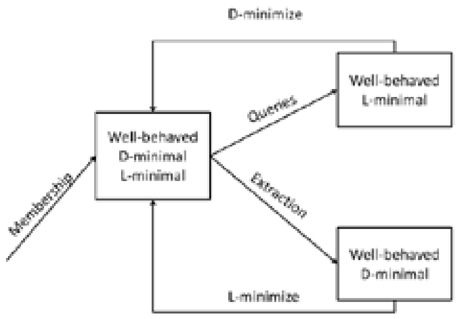
Duplicate Elimination
Consider operator δ applied to relation S with lineage λS. We produce the result R with lineage λ as follows. Note the disjunctive lineage.

(1) For each alternative as in S whose value appears more than once in S, add a single tuple to R with a single alternative a having the same value as as.
(2) Add to R all tuples in S but without any of the alternatives added in step 1.
(3) For each alternative a in R set

$\lambda(a) = \lambda S(s1) \vee \lambda S(s2) \vee \cdots \vee \lambda S(sm)$

where s1, s2, . . . , sm are all the alternatives in S having the same value as a.

Other Operations on ULDB Databases
We have seen how relational queries over ULDB databases are executed. The computation and representation of query answers (though not the possible instances) can depend on whether the input and the output are minimal. we define two notions of minimality for ULDBs: (1) D-minimality, guaranteeing that a ULDB does not contain extraneous data, and (2) L-minimality, guaranteeing that a ULDB does not contain extraneous lineage. We discuss both types of minimization. Because we are tracking lineage, we cannot look at an x-relation in a ULDB in isolation of others. Hence, we consider the extraction problem, where the goal is to



ULDB States and Queries
return only the relation that is the answer to a query (or more generally, a set of x-relations),without the original database. The challenge here is to extract the appropriate lineage along with the result x-relation, so that the correct set of possible instances is preserved. Throughout this section, we confine our attention to ULDB databases that have a certain kind of lineage, called well-behaved lineage. We formally specify well-behaved lineage in above figure summarizes the different operations on ULDBs considered and the possible transitions between states of the ULDB. For instance, the extraction edge guarantees well-behaved lineage and D-minimality are preserved. In order to guarantee well-behaved lineage and L-minimality, the query edge in figure must be restricted to conjunctive queries.

REFERENCE (1) P. Buneman, S. Khanna, and W. Tan. "Why and Where: A Characterization of | Data Provenance". In: Proc. of Intl. Conference on Database Theory (ICDT),2001. | (2) P. Buneman, S. Khanna, and W. Tan. "On Propagation of Deletions and Annotations | Through Views.". In: Proc. of ACM Symp. on Principles of Database Systems (PODS), 2002. | (3) Y. Cui and J. Widom. "Practical Lineage Tracing in Data Warehouses.". In: Proc. of Intl. Conf. on Data Engineering (ICDE), 2000. | (4) Y. Cui, J. Widom, and J. L. Wiener. "Tracing the Lineage of View Data in a Warehousing Environment". ACM Transactions on Database Systems (TODS), Vol. 25, No. 2, 2000.