

DBNotes: A Post-It System for Relational Databases based on Provenance*

Laura Chiticariu
UC Santa Cruz
laura@cs.ucsc.edu

Wang-Chiew Tan
UC Santa Cruz
wctan@cs.ucsc.edu

Gaurav Vijayvargiya
UC Santa Cruz
gaurav@cs.ucsc.edu

1. INTRODUCTION

We demonstrate DBNotes, a Post-It note system for relational databases where every piece of data may be associated with zero or more notes (or annotations). These annotations are transparently propagated along as data is being transformed. The method by which annotations are propagated is based on provenance (aka lineage): the annotations associated with a piece of data d in the result of a transformation consist of the annotations associated with each piece of data in the source where d is copied from. One immediate application of this system is to use annotations to systematically trace the provenance and flow of data. If every piece of source data is attached with an annotation that describes its address (i.e., origins), then the annotations of a piece of data in the result of a transformation describe its provenance. Hence, one can easily determine the provenance of data through a sequence of transformation steps simply by examining the annotations. Annotations can also be used to store additional information about data. Since a database schema is often proprietary, the ability to insert new information about data without having to change the underlying schema is a useful feature. For example, an error report could be attached to an erroneous piece of data, and this error report will be propagated to other databases along transformations, thus notifying other users of the error. Overall, the annotations on the result of a transformation can also provide an estimate on the quality of the resulting database.

Summary of DBNotes Demonstration Features We demonstrate four main features of DBNotes. (1) First, we demonstrate pSQL, an extension of a fragment of SQL that allows one to specify how annotations should propagate through an SQL query. We demonstrate three types of propagation schemes that are supported by pSQL. Annotations can be propagated according to where data is copied from (the default scheme) or according to where data is copied from in *all* equivalent queries (the default-all scheme) or according

to the user specification (the custom scheme). (2) The second feature of DBNotes that we demonstrate is the ability to use pSQL to query both data and its annotations. With this feature, one can pose queries to retrieve, for example, all tuples that are derived from a particular source or all tuples with an attached error report. (3) The third feature of DBNotes that we demonstrate is the ability to provide a detailed explanation on the provenance of an annotation, an attribute value or a tuple in the result of a query transformation. For example, we explain the provenance of a tuple by demonstrating a proof that shows how the query transformation produces the output tuple. We show a binding for each variable in the SQL query and demonstrate that under these bindings, the conditions in the **WHERE** clause are satisfied and the desired output tuple is produced according to the **SELECT** clause of the SQL query. (4) The last feature of DBNotes we demonstrate is its capability to visually describe the journey (i.e., the provenance and flow) taken by a piece of data through various databases. For example, when a user asks about the journey of a piece of data d in a database D , DBNotes displays the sequence of databases and transformation steps that derive d in D . It also displays the sequence of databases and transformation steps that depend on d in D . At the visual interface, the user also has the option of “zooming in” on each transformation step to see a detailed explanation of each transformation step.

To the best of our knowledge, DBNotes is the first Post-It notes system for relational databases that allows a user to specify how annotations should propagate, query annotations and analyze the provenance and flow of data through databases generated by query transformations. We refer the interested reader to [3] for related work.

2. DEMONSTRATION OVERVIEW

We make use of the following example restaurant database for our demonstration.

2.1 Our Example Database

Our example restaurant database contains information about restaurants owned by Hollywood celebrities in the greater Los Angeles area. We have collected information from four different sources [1, 4, 5, 6] and the corresponding schema for each source is shown below:

```
Blue_Pages(Name,Address,Owner,Cuisine,Cost)
LATimes(Res_Name,Location,Owner,Cost,Type_of_food)
Restaurant_Reviews(Name,Owned_by,Food,Address,Expensive)
Seeing_Stars(Name,Address,Getting_there,Owner,Cost,Cuisine)
```

Every attribute value of every tuple may be associated

*Supported in part by an NSF CAREER Award IIS-0347065 and an NSF grant IIS-0430994.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA.
Copyright 2005 ACM 1-59593-060-4/05/06 \$5.00.

Blue_Pages:				
Name	Address	Owner	Cuisine	Cost
Madre's	Pasadena	Jennifer Lopez	Cuban	Expensive

Frequents:		ANNOT(r.Name)
Name	Restaurant	
Ben Affleck	Madre's	a : Blue_Pages:Bad

Figure 1: Bindings for variables r , f and a in Q_2 .

with zero or more annotations. For example, the tuple t_1 : (Madre's {LATimes:Good}, Pasadena, Jennifer Lopez, Expensive, Cuban {May be the worst Cuban food in CA}) in relation LATimes has one annotation attached to the value "Madre's" and one annotation attached to the value "Cuban". The annotations are shown in braces. The rest of the attribute values do not have any annotations. Annotations are not part of the database schema and the method by which annotations are stored and propagated is transparent to the user.

2.2 Propagating Annotations

We first demonstrate a feature of pSQL, an extension of a fragment of SQL, that allows one to specify how annotations should propagate through an SQL query. For example, the pSQL query below integrates information about restaurants from the first two sources and propagates annotations in the default way (i.e., according to where data is copied from). For Q_1 , this means that every tuple in Blue_Pages and LATimes as well as the associated annotations are emitted to the result.

```

Q1:
SELECT DISTINCT Name, Address, Owner, Cuisine, Cost
FROM Blue_Pages
PROPAGATE DEFAULT
UNION
SELECT DISTINCT Res_Name AS Name, Location AS Address,
Owner, Type_of_Food AS Cuisine, Cost
FROM LATimes
PROPAGATE DEFAULT

```

If a tuple occurs in both Blue_Pages and LATimes, only one of the tuples is emitted and the corresponding annotations are unioned together. For example, if t_1 is a tuple in LATimes and t_2 (shown below) is a tuple in Blue_Pages, then the tuple t shown below appears in the output of Q_1 .

t_2 : (Madre's {Blue_Pages:Bad}, Pasadena, Jennifer Lopez, Cuban {Bad Food}, Expensive {Expensive Restaurant})

t : (Madre's {Blue_Pages:Bad, LATimes:Good}, Pasadena, Jennifer Lopez, Cuban {Bad Food, May be the worst Cuban food in CA}, Expensive {Expensive Restaurant})

In addition to the default propagation scheme, we shall also demonstrate the default-all and custom propagation schemes supported by pSQL.

2.3 Querying Annotations

We also demonstrate a feature of pSQL which allows one to query over data and annotations through the following use cases. Suppose we have an integrated view, called Restaurants, of all four sources that is given by executing a query written against Q_1 and the two sources Restaurant_Reviews and Seeing_Stars and the resulting schema is the same as that of Q_1 . An important feature of pSQL that allows annotations to be queried is through the use of the keyword ANNOT(attribute-name) which elevates annotations of

attribute-name to first-class citizens. Some examples of how one could pose queries against data and annotations are shown below in queries Q_2 and Q_3 .

```

Q2:
SELECT DISTINCT f.Name AS Celebrity, r.Name AS Restaurant
FROM Frequents f, Blue_Pages r,
ANNOT(r.Name) a
WHERE f.Restaurant = r.Name AND
r.Cost = 'Expensive' AND
a LIKE '%Blue_Pages:Bad%'
PROPAGATE DEFAULT

```

The query Q_2 retrieves all celebrities that frequent expensive restaurants which received a bad rating from Blue_Pages. (We assume our example database also contains a relation Frequents(Name, Restaurant) that records information about famous stars and the restaurants they visit. This information is obtained from [6].) The *annotation variable* "a" ranges over elements in the set of annotations associated with each Name value of each tuple in Blue_Pages. This query demonstrates how one can query over data and annotations. It also demonstrates how one can query about source information provided that annotations carry information about the sources.

```

Q3:
SELECT DISTINCT r.Name AS Name, COUNT(a) AS Bad_Ratings
FROM Restaurant r, ANNOT(r.Name) a
WHERE a LIKE '%Bad%'
PROPAGATE a TO Name
GROUP BY Name
HAVING COUNT(a) > 2

```

The query Q_3 (with a custom propagation scheme) retrieves bad restaurants where a restaurant is considered as bad if it has more than two bad ratings. This query demonstrates how one can count the number of annotations.

2.4 Explaining Data Provenance

The third feature of DBNotes that we demonstrate is its ability to provide detailed explanations on the provenance of an element (i.e., an annotation, an attribute value or a tuple) in the result of a query transformation. We achieve this in a declarative fashion, by demonstrating a proof that shows how the transformation produced that element.

Consider the query Q_2 in Section 2.3 and let t_3 be a tuple (shown below) in Frequents which states that Ben Affleck is a regular of Madre's. Suppose t_2 (from Section 2.2) is a tuple in Blue_Pages, then the following tuple t' appears in the output of Q_2 :

t_3 : (Ben Affleck {Dated Jennifer Lopez}, Madre's)

t' : (Ben Affleck {Dated Jennifer Lopez}, Madre's {Blue_Pages:Bad})

When asked to provide a detailed explanation of the provenance of the attribute value "Madre's" in the result of Q_2 , DBNotes will first generate a reverse query. The purpose of this query is to extract the relevant source tuples that produced the attribute value "Madre's". DBNotes will show, for each tuple and annotation variable of Q_2 , a binding to source tuples and respectively, source annotations (Figure 1), and demonstrate that under these bindings, the conditions in the WHERE clause are satisfied and the desired output element is produced according to the SELECT clause of Q_2 .

2.5 Tracing the Provenance and Flow of Data

The last feature of DBNotes that we demonstrate is its capability to provide a visual of the "journey" taken by a

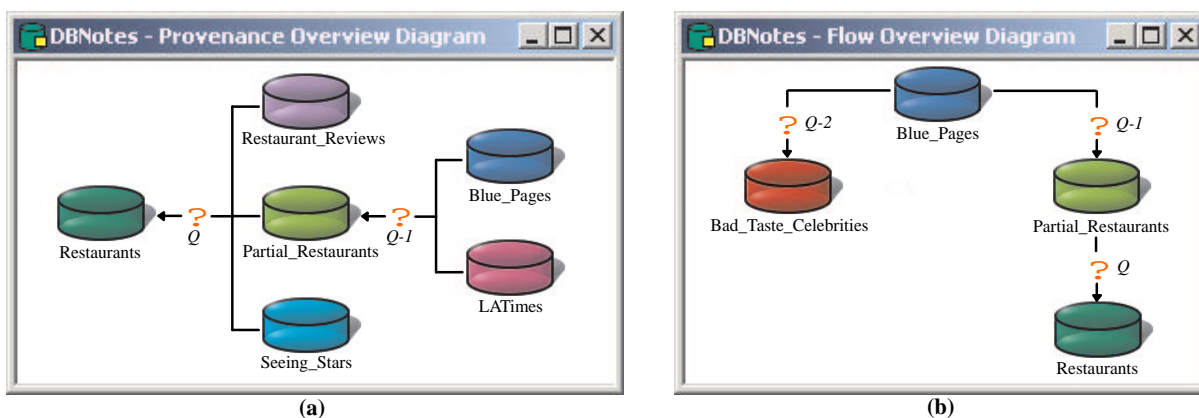


Figure 2: (a) A visual of the provenance of the value “Madre’s” in the relation Restaurants, and (b) a visual of the flow of the value “Madre’s” from the relation Blue_Pages.

piece of data through various databases and transformation steps, where the term “journey” refers to both the provenance and flow of that piece of data. When asked to show the provenance and flow of a piece of data d in a database D , DBNotes shows the sequence of databases and transformation steps that derived d in D as well as the sequence of databases and transformation steps that rely on d in D .

If every value is annotated with its address, then the annotations associated with a value in the result of a query describe the provenance of that value since annotations are, by default, propagated based on provenance. We call such annotations, *provenance annotations*. In fact, DBNotes automatically maintains provenance annotations. Whenever a user writes a query Q over a database D , DBNotes creates a new database based on $Q(D)$ that modifies each annotation by adding the address of each value to each existing annotation associated with that value. To exemplify, suppose the result of query Q_1 is stored in a relation called Partial_Restaurants (“PR” in short). For the provenance of the value “Madre’s” in the integrated view Restaurants, DBNotes displays the diagram shown in Figure 2(a) (assuming query Q integrates information from Partial_Restaurants, Restaurant_Reviews and Seeing_Stars and the latter two also contain information about the restaurant “Madre’s”).

In addition to provenance, DBNotes is also capable of displaying a visual of the flow of a value in a database. The flow of a value, however, cannot be eagerly computed, as we are unaware of what future transformations might depend on the value. Instead, DBNotes computes the flow of a value through a Transformations catalog which records information about source and target databases along with the transformations between them. As an example, when asked to compute the flow of the value “Madre’s” (at address (Blue_Pages, t_2 , Name)) in Blue_Pages of Figure 2(a), DBNotes will show the diagram in Figure 2(b), where the relation Bad_Taste_Celebrities stores the result of query Q_2 . A more detailed description of the mechanisms behind the high-level provenance and flow diagrams of DBNotes can be found in [3].

DBNotes also offers the option of “zooming in” on each transformation step, in order to see a detailed explanation (similar to what was described in Section 2.4) of how a value was produced in the output of that transformation step.

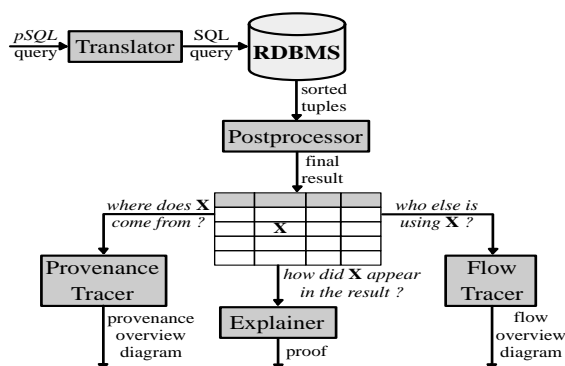


Figure 3: Architecture of DBNotes.

3. SYSTEM ARCHITECTURE

The architecture of our system is illustrated in Figure 3. The explainer module is described in Section 2.4. The provenance and the flow tracer modules are described in Section 2.5. The purpose of the Translator and Postprocessor modules is to compute the result of pSQL queries. The translator module translates a pSQL query into an SQL query against DBNotes’s underlying storage scheme for annotations. The postprocessor module merges together annotations associated with identical addresses. We refer the interested reader to [2] for more details about these two modules. DBNotes is currently implemented with Java v1.4.2 on top of Oracle 9i.

4. REFERENCES

- [1] Restaurant Reviews - Los Angeles Southern California. <http://losangeles.about.com/cs/restaurantreviews/>.
- [2] D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 900–911, Canada, 2004.
- [3] L. Chiticariu, W. Tan, and G. Vijayvargiya. DBNotes: A Post-it System for Relational Databases based on Provenance. Technical Report, University of California, Santa Cruz, 2004.
- [4] Los Angeles Times. <http://www.latimes.com>.
- [5] Online Blue Pages for Restaurants. <http://www.restaurants.com>.
- [6] Seeing Stars in Hollywood. <http://www.seeing-stars.com>.