Lineage Tracing in a Data Warehousing System*

(Demonstration Proposal)

Yingwei Cui and Jennifer Widom
Stanford University
{cyw, widom}@db.stanford.edu

A data warehousing system collects data from multiple distributed *sources* and stores the integrated information as *materialized views* in a local *data warehouse*. Users then perform data analysis and mining on the warehouse views. Figure 1 shows the basic architecture of a data warehousing system.

In many cases, the warehouse view contents alone are not sufficient for in-depth analysis. It is often useful to be able to "drill through" from interesting (or potentially erroneous) view data to the original source data that derived the view data. For a given view data item, identifying the exact set of base data items that produced the view data item is termed the *view data lineage* problem. Motivation for and applications of lineage tracing in a warehousing environment are provided in [2]. In the context of the *WHIPS* data warehousing project at Stanford [3], we have developed a complete prototype that performs efficient and consistent lineage tracing.

Some commercial data warehousing systems support *schema-level* lineage tracing, or provide specialized drill-down and/or drill-through facilities for multi-dimensional warehouse views. Our lineage tracing prototype supports more fine-grained *instance-level* lineage tracing for arbitrarily complex relational views, including aggregation. Our prototype automatically generates *lineage tracing procedures* and supporting *auxiliary views* at view definition time. At lineage tracing time, the system applies the tracing procedures to the source tables and/or auxiliary views to obtain the lineage results and show the specific view data derivation process.

1 Lineage Tracing System

1.1 Lineage Example

Given a view data item I, the exact set of source data that produced I is called I's *lineage*. We use an example to illustrate the concepts; a full formalization of the problem along with solutions and algorithms are given in [2]. Consider a financial data warehouse with the three source tables shown in Figure 3. A view `Promising` (Figure 4) is defined to contain all "promising" industries, where an industry is regarded as promising if some stock in that industry is gaining money over all purchases, and the stock has a price-earnings ratio below 40. Over our sample source data the view contains two tuples, ⟨`computer`⟩ and ⟨`medicine`⟩. To learn more about why tuple ⟨`computer`⟩ is in the view, the user may choose to trace its lineage. The result is shown in Figure 6.

1.2 Tracing Procedure Generation

In general, to compute the lineage of a view data item, we need the view definition and the original source data, and perhaps some auxiliary information. (In Section 1.3 below, we describe how we can perform lineage tracing without access to the source data.) In our prototype, we first transform the view definition into a normal form composed of aggregate-project-select-join sequences, called *ASPJ segments*. The lineage of tuples in a view defined by a single ASPJ segment can be computed using relational queries over the sources, called *tracing queries*, which are parameterized by the tuple(s) being traced. To trace the lineage of a view defined by multiple levels of ASPJ segments, we logically define an *intermediate view* for each segment, and recursively trace through the hierarchy of intermediate views top-down. At each level, we use tracing queries for a one-level ASPJ view to

*This work was supported by DARPA and the Air Force Rome Laboratories under contracts F30602-95-C-0119 and F30602-96-1-0312.

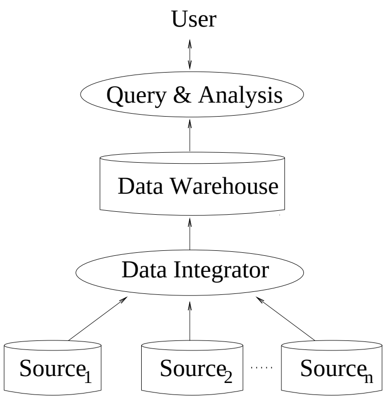

Figure 1: Basic warehousing architecture

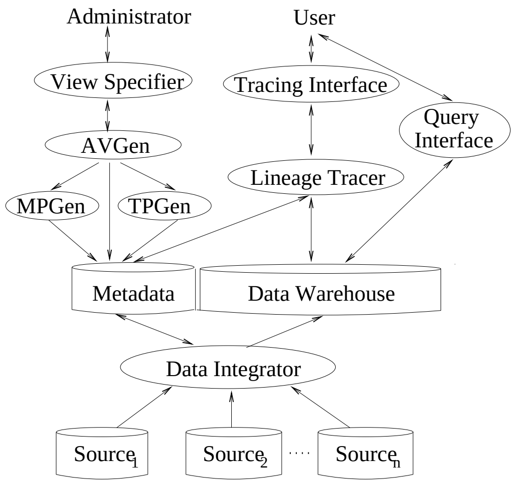

Figure 2: The lineage prototype

Daily

ticker	high	low	closing
AAA	29	25	26
BBB	89	87	89
CCC	75	74	74
DDD	120	100	120

Earnings

ticker	industry	earnings
AAA	automobile	0.5
BBB	computer	4.0
CCC	computer	1.2
DDD	medicine	2.0

Purchases

ticker	date	price	shares
BBB	1/5/98	16	100
CCC	3/24/98	80	300
BBB	6/7/98	40	50
DDD	6/11/98	80	200

Figure 3: Sample source data

```
CREATE VIEW Promising AS
SELECT e.industry
FROM Purchases p, Daily d, Earnings e
WHERE p.ticker = d.ticker
  AND d.ticker = e.ticker
  AND d.closing/e.earnings < 40
GROUP BY p.ticker, e.industry
HAVING
  SUM(p.price*p.shares)/SUM(p.shares) < d.closing
```

Figure 4: View `Promising`

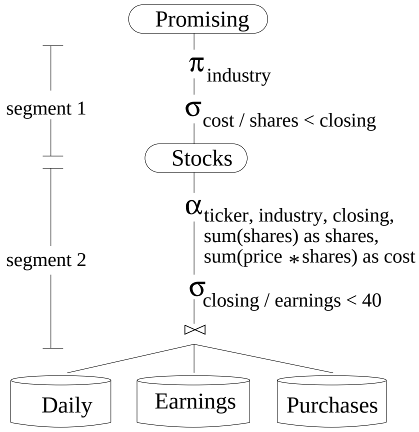

Figure 5: Normalized view definition

Daily

ticker	high	low	closing
BBB	89	87	89

Earnings

ticker	industry	earnings
BBB	computer	4.0

Purchases

ticker	date	price	shares
BBB	1/5/98	16	100
BBB	6/7/98	40	50

Figure 6: Lineage of ⟨`computer`⟩ according to `Promising`

compute the lineage for the current traced tuples with respect to the views or base tables at the next level below.

For example, consider the view `Promising` in Figure 4. Figure 5 shows `Promising`'s normalized view definition with two ASPJ segments and one intermediate view `Stocks`. We generate one tracing query for segment 1, where `t` is the tuple to be traced:

```
SELECT * FROM Stocks WHERE industry = t.industry AND cost/shares < closing
```

For segment 2 we have three tracing queries, and we trace a tuple set `T`:[1]

```
SELECT * FROM Daily WHERE ticker IN (SELECT ticker FROM T)
SELECT * FROM Earnings WHERE ticker IN (SELECT ticker FROM T)
SELECT * FROM Purchases WHERE ticker IN (SELECT ticker FROM T)
```

When tracing the lineage of tuple $t = \langle \texttt{computer} \rangle$ in `Promising`, we first use the tracing query for segment 1 to compute t's lineage in `Stocks`; the result is $\texttt{Stocks}^* = \{\langle \texttt{BBB}, \texttt{computer}, 89, 150, 3600 \rangle\}$. We then trace the lineage of $\texttt{Stocks}^*$ in the source tables using the three queries for segment 2, obtaining the final result as shown in Figure 6. Details and optimizations appear in [2].

1.3 Auxiliary View Generation

In the example of Section 1.2 we introduced an intermediate view `Stocks` for the purpose of lineage tracing. In general, such intermediate views can either be materialized, or we can recompute the relevant intermediate results at tracing time. Because efficient incremental maintenance of multi-level aggregate views generally requires materializing the same intermediate views we need for lineage tracing [4], we always choose to materialize these views as *auxiliary views* in the warehouse. A second type of auxiliary view is motivated by the fact that in a distributed multi-source data warehousing environment, querying the sources for lineage information can be difficult or impossible: sources may be inaccessible, expensive to access, and/or inconsistent with the views at the warehouse. By storing additional auxiliary views in the warehouse based on the source tables, we can reduce or entirely avoid source accesses for lineage tracing. There are numerous options for which auxiliary views to store, with performance tradeoffs; see [1] for details. All of the auxiliary views we create to support lineage tracing are maintained consistently with the user views in the warehouse by the WHIPS prototype [3].

1.4 Prototype Architecture and Implementation

In Figure 2 we expand the *Query & Analysis* component in Figure 1 to illustrate the architecture of our lineage tracing prototype. When a view is defined through the *View Specifier*, if the view definition specifies that the view should be traceable, then the *Auxiliary View Generator* (*AVGen*) automatically generates the auxiliary views discussed in Section 1.3. The *Maintenance Procedure Generator* (*MPGen*) and the *Tracing Procedure Generator* (*TPGen*) then generate the maintenance procedures (see [3]) and lineage tracing procedures for the user view as well as its auxiliary views, and store them as part of the *Metadata*. When a user issues a request through the *Tracing Interface*, the *Lineage Tracer* is activated and calls the appropriate sequence of tracing procedures. The lineage results are then returned to the user as tables. If the user further requests to see the *derivation process*, the lineage tracer combines the lineage results and the view definition to generate a *derivation tree* for the user, showing the complete lineage information as in Figure 7 below.

2 Demo Description

Our demonstration will be based on a financial warehouse scenario with three source tables and several views, some quite complex. A web-based GUI interface is provided to interact with the warehouse and the lineage tracing system. We will illustrate the various capabilities of the tracing system. A screen shot showing the complete lineage, including derivation process, for a tuple in one of our views is shown in Figure 7. Note that as of July 1999 the system is fully implemented and functional.

[1] The tracing queries in this example are simple, but they can become quite complex in the general case [2].

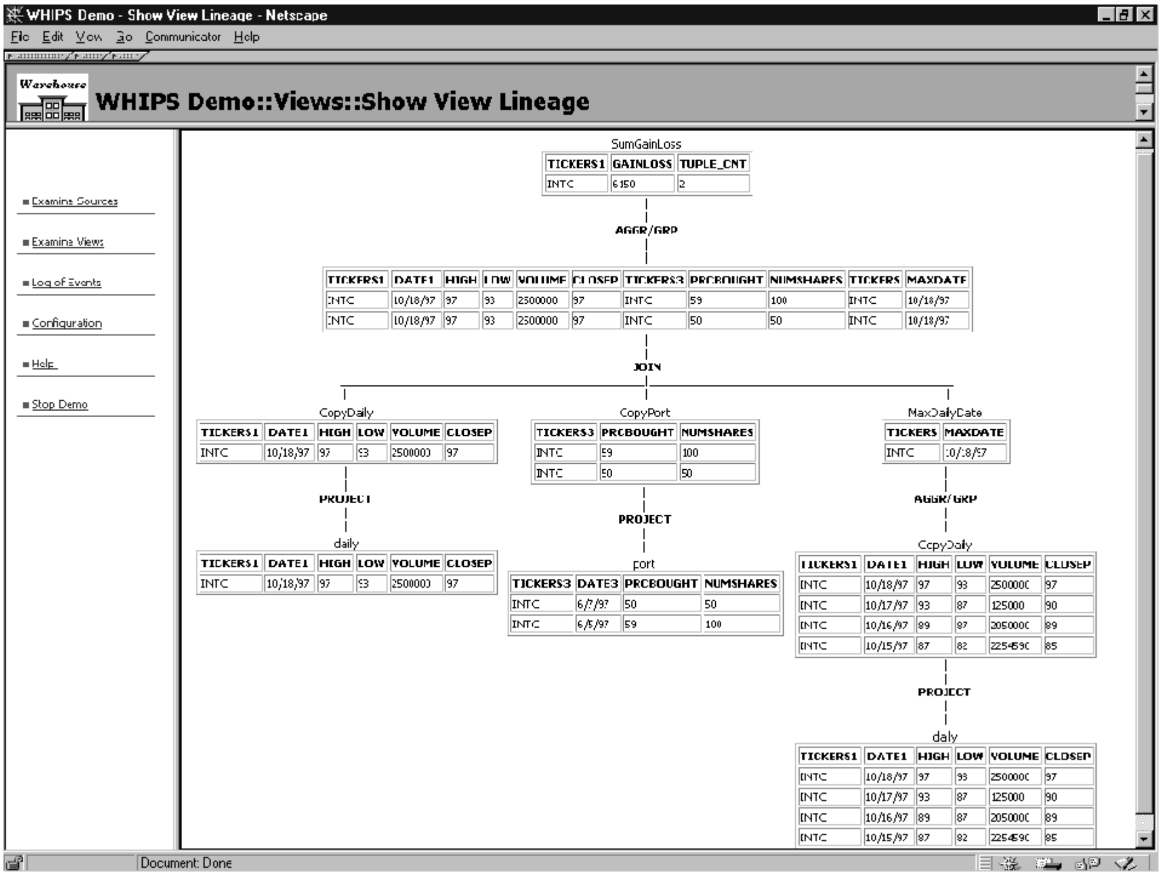

Figure 7: The demo interface

References

[1] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. Technical report, Stanford University Database Group, January 1999. Available at http://www-db.stanford.edu/pub/papers/trace.ps.

[2] Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a warehousing environment. Technical report, Stanford University Database Group, November 1997. Available at http://www-db.stanford.edu/pub/papers/lineage-full.ps.

[3] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):41–48, June 1995.

[4] D. Quass. Maintenance expressions for views with aggregation. In *Proc. of the Workshop on Materialized Views, Techniques and Applications*, pages 110–118, Montreal, Canada, June 1996.