Administering Propagated Metadata in Large, Multi-Layer Database Systems[1]

Arnon Rosenthal
The MITRE Corporation
arnie@mitre.org

Edward Sciore
Boston College and MITRE
sciore@bc.edu

Abstract

Enterprise databases are comprised of multiple local databases that exchange information. The component databases will rarely have the same native form, so one must map between the native interfaces of the data suppliers and the recipients. An SQL view is a convenient and powerful way to define this map, because it provides not just an evaluation mechanism, but also query, and (to some degree) update and trigger capabilities. However, SQL views do not map the critical metadata (e.g., security, source attribution, and quality information) between data suppliers and their recipients.

This paper examines the research problems arising from the creation of a metadata propagation framework: a theory for inferring and reconciling metadata on views, rules for each property type and data derivation operator, efficient implementation, and ways to coordinate metadata administration across multiple schemas.

1. Introduction

1.1. The Problem

A large database system is likely to involve large amounts of metadata (e.g., data quality measures, integrity constraints, security information, or schema documentation). Metadata helps users understand the data, and helps administrators plan improvements and schema changes. Metadata is costly to obtain and maintain. Administrators must determine which metadata to solicit, and then convince data providers to supply that metadata with their data. It is vital to minimize the maintenance burden.

A large database system is also likely to be *multi-layer* – that is, composed of several different schemas, with physical (source) schemas at the bottom and virtual (view) schemas layered on top. Multi-layer databases take many forms. For example, a set of view tables can be used to insulate applications from stored tables that have been partitioned or denormalized, and which change as the workload changes. A federated database provides a virtual schema above multiple sources. A data warehouse gathers and transforms data and stores it in a separate server; this can be seen as computing a materialized view (subject to delays in propagating source updates). Web-oriented distributed systems often have layers of objects derived from each other. The specification of a virtual schema can be as simple as a "create view" SQL statement, or as complex as a series of warehouse transformations involving data scrubbing, integration of multiple sources, and multiple levels of aggregation over several dimensions.

Our research begins with the observation that existing schema transformation algorithms and tools propagate data from source layers to view layers, but do not propagate metadata. That is, all of the metadata that was painfully collected at a source schema is "locked" into that schema; users of virtual schemas see none of it. The goal of our work is to create a virtuous circle. When metadata can be exploited more widely, it will have greater value to the organization, leading to incentives to capture more metadata and to propagate it more widely. The result is that databases will be better described and more usable.

We therefore investigate how to propagate metadata between schemas. Three challenges must be met:

- *Diverse propagation semantics:* For example, one might define the access rights associated with a join result to be the *intersection* of the sources' access rights, and the trustworthiness to be the *minimum* of the sources' trust values.
- *Scale:* Each warehouse data attribute might have ~10 metadata properties (e.g., ownership, responsibility, precision, units, accuracy, timeliness, …). Administrators simply lack the time to extend views' attribute derivations to derive all these meta-attributes.
- *Coordination:* No schema is an island. When planning a change, the effects on all relevant schemas must be considered, and several administrators may need to participate.

[1] appeared at IEEE Workshop on Knowledge and Data Exchange, 1999., P. Scheuermann (ed.)

All three difficulties can be approached through inference rules that generate assertions about view metadata (based on the sources' metadata and the derivation function). Section 2 presents our approach to inference rules. Section 3 then considers the issue of how to support coordination of work across different schemas. Section 4 contains conclusions and open research problems. Reference [1] contains an initial mockup (pure html) of how one might capture and use inference rules.

2. Metadata Capture and Inference

2.1. Preliminaries

Metadata assertions take the form *(database granule, property, value).* An asserted property value denotes that the granule's property *dominates* (is better or equal to) the bound. A database granule is a subset of a database, e.g., a table, column, row, cell, or view. A *property* describes some meta-information about the granule, e.g., credibility, error-bound, access permission, timestamp, and so forth.

Each property takes values from a *domain.* The domain must be partially ordered by "better than"; better metadata indicates some property of the data is superior. "Better" often corresponds to numeric > (e.g., for credibility), numeric < (e.g., for error bound), or set containment (e.g., "accessible to {joe, fred, sue}" is better than "accessible to {joe}"). To make it easy to combine information, we generally require that the domain have top and bottom values (±∞) and a *summary operator* , a *unique* least upper bound operator that returns the least value that dominates its inputs (e.g., max credibility, min error bound, or union of two access lists). With this approach, as long as asserted metadata is correct and inference (discussed below) is sound, we have a consistent system.

Metadata is most commonly used to denote lower bounds, i.e., assertions that denote that a value is "good enough". It is also possible to express upper bounds, if desired. Upper bounds give rise to a dual theory, with a separate set of inference rules, and are not discussed further in this paper.

Metadata can be asserted directly on derived granules (i.e., on views), as well as on base tables. For example, recent data (e.g., SALES where Year>1998) may have better credibility and precision than the overall SALES information, and summarized information (e.g., Total SALES.Amt, Dept# grouped by Dept#) may have a more liberal security policy.

All assertions are considered correct, whether directly asserted or provided by an inference rule. One can then obtain a valid bound by taking the of the bounds obtained from all such values. A system can obtain bounds on view metadata from three sources: bounds asserted explicitly on the view, bounds inferred from the view's definition, and bounds inferred from other queries that are known to be equivalent to the view query.

2.2. Inference Rules

2.2.1. Theory of Inference Rules. Inference rules are functions that produce assertions on view metadata from the metadata asserted on sources. The rules will be provided by metadata tool vendors (for generic rules) or by sophisticated administrators (for organization-specific rules).

Rule providers must be cautious, as we insist that rules be *sound.* That is, in every case where the rule is asserted to apply, it yields a correct bound (though not necessarily the best possible one). The motivation is that sound rules can be composed without knowledge of the particular context. In contrast, it seems too hard to interpret values inferred by a sequence of rules each of which "is usually true" or "gives a good approximation".

Inference rules will be illustrated using a running example. Let RECENT_SALES be the view (SALES where Year > 1998), and suppose that the table SALES has properties *Accuracy* (defined as the likelihood that a randomly-chosen cell's value will be correct), and *Precision* (defined as the number of digits that are significant, worst case, for all cells).

In principle, one might need a separate inference rule for computing each property of each view granule. For example, suppose we were certain that in our company, data collection practices have improved. Then we might have a rule that lets us infer that a bound on SALES.Accuracy applies also to RECENT_SALES.Accuracy. Such a rule would then be reevaluated when the inputs' metadata changed.

For access permissions, SQL99 semantics use only direct assertions, except that the view owner's privileges are inferred from permissions on the sources. In [2], we showed that one can obtain much greater power if the query processor is able to identify alternative expressions.

2.2.2. Tractable Cases – Obtaining Inference Rules En Masse. A full set of inference rules would provide a rule for inferring each property for each granule. This would be an enormous amount of work, often requiring both external world knowledge and programming skills. There are cases (e.g., determining the precision of a value that is the result of a multiplication) where determining an appropriate rule is nontrivial. Fortunately, the complexity of the general case is not always necessary. This section identifies situations where one can determine rules more easily. If we were building a commercial tool tomorrow, we would incorporate them.

The goal is to have rules that apply to many granules, and are provided by software vendors. Such rules must rely on semantics of inferred property and of the data derivation operators. (General views can be seen as expression trees, and handled stepwise from the leaves.) Even an incomplete set of inference rules can provide a great deal of metadata that is unavailable today; we would be satisfied with partial success. We thus explore three tractable situations: trivial derivations, instance properties that bound the worst case, and query-insensitive properties.

First, many views perform trivial mappings, such as attribute renaming, units conversion (multiplication), and some format transforms. While technically trivial, such mappings can occur hundreds of times in a federated schema, either standalone or as part of more complex views. Hence automating even trivial propagations can be a real benefit.

In cases we have seen, all properties except "language" were invariant under renaming. Under constant multiplication (e.g., for units conversion), many properties (e.g., security, credibility) are invariant, and others simply scale (e.g., precision). Sometimes more complex rules can be obtained [3], e.g., for precision of a scalar expression, based on precision of the inputs.

Second, many properties describe *instance* properties – a value that bounds every cell in the database granule described (e.g., precision, worst case error, "Joe has permission to read"). Such a bound obviously applies to any subset of those cells. Hence, the bound for the source granule is also correct for the view (though not necessarily tight). The inference rule for such views is thus: `identity`. Such a rule applies to `Project`, and to both ordinary `Select` and `Select` with a nested subquery.

Finally, for many properties, the view metadata can be bounded as a function of the inputs' metadata, regardless of what operators are used in the view. We call such rules *query insensitive.* They generalize the single inference rule that [4] applies to several kinds of *Boolean* metadata.

In the examples below, query insensitive lower bound rules can be expressed by applying an associative, commutative *constructor operator* (usually, `greatest lower bound`) to the sources' metadata values. Five query-insensitive properties, with their constructor functions, are:

- *Security permissions*. *Security permissions*. One has the right to execute a query if one has the right to access each source table. If each permission is represented by the access predicate p_i, then the bound inferred for the view is AND(p_i).
- *Timestamps*. Suppose each source table has a timestamp meta-attribute that means "this data is guaranteed up to date, as of this timestamp". Then the view meta-attribute infers the worst (i.e., the minimum) timestamp of any of its inputs.
- *Integrity level.* Data may be kept at several integrity levels, representing successive levels of care (e.g., raw, reviewed, confirmed, warranted…) [5, ch. 2.8]. The inferred bound is the worst level of any of its source values.
- *Source attribution.* Each source table has a meta-attribute listing the origins of its data. The view table's attribution is inferred as the union of the source lists.
- *Pricing.* Suppose each input table has a meta-attribute giving the cost of accessing it. Then the cost associated with the view is at most the sum of these costs.

2.2.3 Top-Down Propagation (Update to Derived Metadata). Inference rules drive *bottom-up propagation*. That is, bounds on views' property values are determined from the bounds on the views' input properties. The combined, best-available bound can be considered as a view over the relevant directly-asserted metadata (comprising assertions on both source and view granules).

Top-down propagation refers to attempts to change the combined bound on view metadata. Like updates to ordinary data views, the goal is to change the sources to change the derived value. This is different from simply asserting a new bound. First, the desired new value may be inconsistent with sources—worse that the bound inferred—so the source assertions need to be changed (lowered). Even if the new value is consistent (i.e above the inferred lower bound), one has the choice between providing a new assertion on the view, or an assertion about source metadata that will give the desired view behavior.

For example, suppose a view administrator requests that the role DataMiner be given access to RECENT_SALES. A source administrator can achieve this by simply granting permission on the view; alternatively, if the date is irrelevant to the decision and further needs are anticipated, one might prefer to grant more generally, e.g., on SALES, or on SALES where Date>1990.

Thus, top down propagation (like ordinary view update) tends to be semantically ambiguous. The only theoretical guidance is that a legal implementation should achieve the desired effect on the view. Beyond that, it will often be necessary for a human to intervene to determine which course to take, as discussed in Section 3.

3.0. Communicating about Metadata

This section addresses the tension between two forces on metadata administrators:

- Metadata updates are naturally formulated or judged in terms of the currently-visible schema.

- No schema is an island, since inference rules propagate the updates.

3.1. Propagating a Change: Actions and Scope

Cross-schema coordination of updates to metadata involves more than ACID-transaction consistency; equally important is knowing the desirability of the changes to the various stakeholders, and helping them to reach agreement. For example, an information provider may agree to consult with consumers who use views, before making changes that can reduce consumers' access or data quality. This tension gives rise to the research problem: *Devise primitives for coordinating metadata updates among tables related by data derivation.*

A cursory examination reveals that change requests have many modes, e.g., commands, notifications, and "trial balloons" that are posted to see user reaction. Our first thought was to express each as a different property, so one could see both the current and proposed value. However, it is not tolerable to similarly expand the set of inference rules. We therefore need a model that allows a single rule to apply to metadata that differ only in the mode of its assertion.

In the next subsection, we give several examples to provide motivation and requirements. In addition, we show how the situations would be represented in our (first cut) model for separating "desired *action*" from "property value being considered". In this model, a request to change a meta-attribute instance G.M can take several forms. The ones we illustrate are:

- *Command* that the specified value of G.M be installed in the database.
- *Propose* that the specified value of G.M be installed in the database
- *Request* the recipient to make the change, without having authority to insist.
- *Notify* others of the current value of G.M.

Each of these contexts is called an *action*. The scenarios below show that inference rules can be insensitive to the action; we hypothesize that this is true in general.

We next observe that one may not wish to broadcast the change to the world, when starting a discussion. Hence one needs the ability to specify who should be informed, i.e., a *scope*. The scope helps us send a description of the change, expressed in terms of the right schema, to the right agent (human or an automated service). An action can have a scope of either *explicit* (private to recipients listed in the message) or *unlimited* (any user to whom effects could propagate).

We envision a loose coupling, in which received values do not immediately overwrite the current ones. Local facilities would process the incoming messages to apply the received values appropriately, e.g., by notifying humans immediately, or batching for later attention, or by calling installation scripts. Section 4.2 speculates about further extensions in this vein.

3.2. Example Scenarios

This section considers several example scenarios. For each scenario we describe the implied action, its scope, and the effect of the propagation. For top-down propagation, semantic ambiguity is resolved by propagating several proposals, for humans to handle. Further features are needed to allow recipients to indicate acceptance, and to have an appropriate Commit protocol for coordinated decisions.

These scenarios assume that the required inference rules do exist, and that for top-down propagation the software has generated proposals for a reasonable set of semantic alternatives.

The five scenarios below all address a system with source databases, a warehouse, and a federation layer. An Information Systems group, denoted *IS*, manages source databases (relational); each database has its own administrator. There are also two derived databases, each controlled by a user department. One user department extracts information into a data warehouse, and processes requests entirely in terms of this warehouse. The other department creates a federated schema. Its user requests are treated as requests against a view, and translated automatically into programs that access the source databases. The same set of user roles (e.g., Actuary) are known to all three systems.

- The IS dept has established that only certain user groups (e.g., Actuaries, ClaimsAdjusters) may access information whose source was IS data. The warehouse-keepers need to infer what permissions are allowable for the warehouse to grant. These grants could be executed automatically by the warehouse, or the warehouse could treat them as an upper bound and make manual assignments. Federation users will find that permissions are checked against source tables, but still want to know what will be allowed. Hence they too use the inferred permissions.

  From the source to the warehouse, this is a command action (scope=warehouse). That is, it is obligatory to keep warehouse assertions consistent with the inferred values (though they can be lower). For (scope= federation) views, it is a Notify action. The IS department recommends but does not require that the federation's informative permissions be in synch.
- The IS dept is considering revoking some access permissions on its source tables, but wants to ask whether view parties have strong objections. The view parties see the proposed changes (in terms of their view tables) and respond.

This is a propose message, propagated bottom-up from IS. Note that the source uses propose messages to do “what if” analysis. Its scope might be global. However, only users that IS has contracted to support may have a right to protest.

- A warehouse administrator sends a message to the IS dept requesting additional write permissions on the RECENT_SALES view. The propagation mechanism generates two possible interpretations: grant additional permissions on the view, or grant additional permissions on the base table. These interpretations are received by the IS dept, which can choose either route. Alternatively, they can counter-propose, e.g., to give permission on SALES where DATE>1990, or even to grant only part of the desired view privilege, e.g., RECENT_SALES where Amount<100K.

  Both the initial message and any counter-proposals are propose commands, explicit scope. The initial message is propagated top-down. When both parties are satisfied, the IS dept will send a global command message describing the changes made. As illustrated, the parties can define new views and counter-propose permissions in terms of those views.

- The company wants to compare the warehouse’s enforced permissions with the allowed permissions, to identify excessive grants at the derived schemas, and also to show source permissions that seem not to be in use. The auditors translate the permissions so they are expressed against the source schema, and then compare them.

  This is a notify message, global scope, with inference bottom-up from IS. A variant is to have the test initiated from one view schema, e.g., the warehouse. The inference is still bottom up, but propagation would be explicitly scoped (i.e., to the warehouse).

4. Summary and Open Problems

4.1. Benefits

We have provided a conceptual framework for meta-attribute propagation and communication in multi-layer database systems. This area has received little attention in academia or industry, perhaps because it is difficult to provide services that span DBMSs, organizations, and approaches to metadata semantics and representation. For administrators, our basic goal is to allow each administrator to work within their own schema, without tracing through derivation logic. Automating even a part of this task would make much more metadata available for the various schemas, and greatly aid negotiators.

Our approach is based on asserting bounds on metadata. This provides robustness. The inference system shows how one can infer among diverse schemas, and combine evidence from multiple sources. A query rewriter can automatically generate queries equivalent to a view definition. Incomplete metadata information, an incomplete set of inference rules, and incomplete enumeration of alternative computation paths may result in a looser bound, but it will still be correct. This graceful degradation makes the theory seems very appropriate for wizards that assist administrators.

We showed the requirements for collaboration, with a variety of scenarios. Our collaboration categories give useful constructs for guiding what should be done with inferred metadata. They enable a loosely coupled architecture, where arriving metadata is stored into ancillary objects, separate from the current values. Local mechanisms can then apply them appropriately (e.g., to notify humans, or invoke installation scripts).

4.2. Some Open Problems

Research is needed to develop a more comprehensive solution. The system metadata management environment should split roughly into modules that do propagation, plus those that do coordination.

For propagation, there are many open problems in defining and managing inference rules. First, for nearly every kind of metadata and every data derivation operator, *sound* inference rules are needed. For example, what is the precision of a scalar product or a column total (in terms of its inputs’ precision and the number of rows); given various assumptions about foreign keys, what are the completeness and credibility of a join and semijoin? Each case is a narrow research task, well suited to beginners. A tough, general problem is to find a use for defaults, or “approximate” or “usually accurate” inference rules. Next, one needs to configure query processors to reveal alternative expressions for a query. One then needs to produce algorithms that efficiently find and apply relevant rules, exploiting simplifications (e.g., where and form a lattice) but handling realistic complications (full query languages, and sets of tables that are mutually derivable).

We do not yet understand how to modularize a system for such distributed metadata. Experimentation is needed. Inference rules (which capture much metadata semantics) may also have a part to play in controlling collaboration.

For coordination, the facilities need to be extended, to reduce burdens on administrators. Currently, an administrator must specify the scope for propagation, and the mode of using the propagated values at their destinations. For top down propagation, the administrator has the additional burden of managing sets of alternative translations. Finally, currently administrators must manually determine appropriate scope, and the mode in which the recipient should treat received values.

5. References

[1] A. Rosenthal, E. Sciore, G. Gengo, “Demonstration of Multi-Tier Metadata Management,”at http://www.cs.bc.edu/~sciore/papers/demo.zip

[2] A. Rosenthal and E. Sciore, “Security Administration for Federations, Warehouses, and other Derived Data.” *IFIP 11.3 Working Conference on Database Security,* Seattle, WA, 1999 http://www.cs.bc.edu/~sciore/papers/secadmin.ps

[3] H. Kon, *Data Quality Management: Foundations in Error Measurement and Propagation*. Ph.D. Thesis, MIT Sloan School of Management, 1996,

[4] A. Motro, “*Panorama: A Databases System that Annotates Its Answers to Queries with their Properties*,” J. Intelligent Info. Systems, 1996.

[5] S. Castano et. al., *Database Security*, Addison-Wesley, 1994.