Proc. of 4th Int'l Conference on Information and Knowledge Management (CIKM), Nov. 1995

Research Problems in Data Warehousing

Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305-2140
widom@db.stanford.edu

Abstract

The topic of data warehousing encompasses architectures, algorithms, and tools for bringing together selected data from multiple databases or other information sources into a single repository, called a *data warehouse*, suitable for direct querying or analysis. In recent years data warehousing has become a prominent buzzword in the database industry, but attention from the database research community has been limited. In this paper we motivate the concept of a data warehouse, we outline a general data warehousing architecture, and we propose a number of technical issues arising from the architecture that we believe are suitable topics for exploratory research.

1 Introduction

Providing integrated access to multiple, distributed, heterogeneous databases and other information sources has become one of the leading issues in database research and industry [6]. In the research community, most approaches to the data integration problem are based on the following very general two-step process:

1. Accept a query, determine the appropriate set of information sources to answer the query, and generate the appropriate subqueries or commands for each information source.

2. Obtain results from the information sources, perform appropriate translation, filtering, and merging of the information, and return the final answer to the user or application (hereafter called the *client*).

We refer to this process as a *lazy* or *on-demand* approach to data integration, since information is extracted from the sources only when queries are posed. (This process also may be referred to as a *mediated* approach, since the module that decomposes queries and combines results often is referred to as a *mediator* [20].)

The natural alternative to a lazy approach is an *eager* or *in-advance* approach to data integration. In an eager approach:

1. Information from each source that may be of interest is extracted in advance, translated and filtered as appropriate, merged with relevant information from other sources, and stored in a (logically) centralized repository.

2. When a query is posed, the query is evaluated directly at the repository, without accessing the original information sources.

This approach is commonly referred to as *data warehousing*, since the repository serves as a warehouse storing the data of interest.

A lazy approach to integration is appropriate for information that changes rapidly, for clients with unpredictable needs, and for queries that operate over vast amounts of data from very large numbers of information sources. However, the lazy approach may incur inefficiency and delay in query processing, especially when queries are issued multiple times, when information sources are slow, expensive, or periodically unavailable, and when significant processing is required for the translation, filtering, and merging steps. In cases where information sources do not permit ad-hoc queries, the lazy approach is simply not feasible.

In the warehousing approach, the integrated information is available for immediate querying and analysis by clients. Thus, the warehousing approach is appropriate for:

- clients requiring specific, predictable portions of the available information

- clients requiring high query performance (the data is available locally at the warehouse), but not necessarily requiring the most recent state of the information

- environments in which native applications at the information sources require high performance (large multi-source queries are executed at the warehouse instead)

- clients wanting access to private copies of the information so that it can be modified, annotated, summarized, and so on, or clients wanting to save information that is not maintained at the sources (such as historical information)

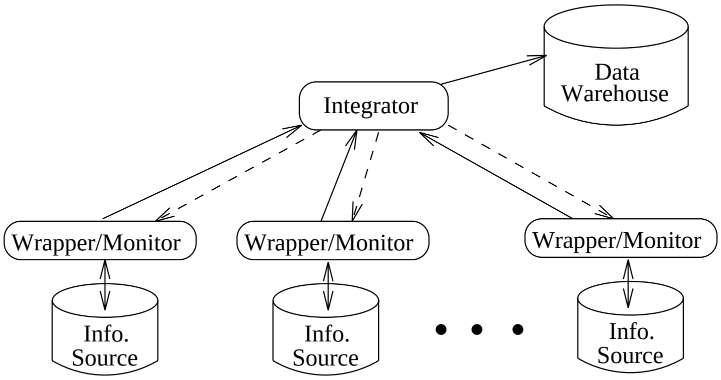

Figure 1: Basic architecture of a data warehousing system

The lazy and warehousing approaches are each viable solutions to the data integration problem, and each is appropriate for certain scenarios.[1] The database research community has focused primarily on lazy approaches to integration. In this paper we consider research problems associated with the warehousing approach.

2 Industrial Perspective

Before considering the research problems associated with data warehousing, we note that there has been great interest in the topic within the database industry over the last several years [12]. Most leading vendors claim to provide at least some "data warehousing tools," while several small companies are devoted exclusively to data warehousing products. Despite rapid advances in commercial data warehousing tools and products, most of the available systems are relatively inflexible and limited in their features. We believe that a truly general, efficient, flexible, and scalable data warehousing architecture requires a number of technical advances, outlined below.

The importance of data warehousing in the commercial segment appears to be due to a need for enterprises to gather all of their information into a single place for in-depth analysis, and the desire to decouple such analysis from on-line transaction processing systems. Analytical processing that involves very complex queries (often with aggregates) and few or no updates—usually termed *decision support*—is one of the primary uses of data warehouses, hence the terms data warehousing and decision support often are found together, sometimes interchanged.[2] Since decision support often is the goal of data warehousing, clearly warehouses may be tuned for decision support, and perhaps vice-versa. Nevertheless, decision support is a very broad area, so we focus this paper specifically on research issues associated with the warehousing approach to integration.

3 Architecture of a Data Warehousing System

Figure 1 illustrates the basic architecture of a data warehousing system. The bottom of the diagram shows the information sources. Although the traditional disk shapes connote conventional database systems, in the general case these sources may include non-traditional data such as flat files, news wires, HTML and SGML documents, knowledge bases, legacy systems, and so on. Connected to each information source is a *wrapper/monitor*. The *wrapper* component of this module is responsible for translating information from the native format of the source into the format and data model used by the warehousing system, while the *monitor* component is responsible for automatically detecting changes of interest in the source data and reporting them to the integrator.

When a new information source is attached to the warehousing system, or when relevant information at a source changes, the new or modified data is propagated to the *integrator*. The integrator is responsible for installing the information in the warehouse, which may include filtering the information, summarizing it, or merging it with information from other sources. In order to properly integrate new information into the warehouse, it may be necessary for the integrator to obtain further information from the same or different information sources. This behavior is illustrated by the downward dashed arrows in Figure 1.

The data warehouse itself can use an off-the-shelf or special purpose database management system. Although in Figure 1 we illustrate a single, centralized warehouse, the warehouse certainly may be implemented as a distributed database system, and in fact data parallelism or distribution may be necessary to provide the desired performance.

The architecture and basic functionality we have described is more general than that provided by most commercial data warehousing systems. In particular, current systems usually assume that the sources and the warehouse subscribe to a single data model (normally relational), that propagation of information from the sources to the warehouse is performed as a batch process (perhaps off-line), and that queries from the integrator to the information sources are never needed.

[1] Another promising and relatively unexplored approach to information integration is a *hybrid* approach, in which some information is stored in a centralized repository while other information is fetched on demand, e.g., [21].

[2] Other relevant terms include *data mining*, *on-line analytical processing* (*OLAP*), and *multidimensional analysis*, which we view as refinements or subclasses of decision support.

4 Research Problems

Based on the general architecture for data warehousing described in Section 3, we now outline a number of research problems that arise from the warehousing approach.

4.1 Wrapper/Monitors

The *wrapper/monitor* components illustrated in Figure 1 have two interrelated responsibilities:

1. **Translation:** Making the underlying information source appear as if it subscribes to the data model used by the warehousing system. For example, if the information source consists of a set of flat files but the warehouse model is relational, then the wrapper/monitor must support an interface that presents the data from the information source as if it were relational. The translation problem is inherent in almost all approaches to data integration—both lazy and eager—and is not specific to data warehousing. Typically, a component that translates an information source into a common integrating model is called a *translator* or *wrapper* [3, 20].[3]

2. **Change detection:** Monitoring the information source for changes to the data that are relevant to the warehouse and propagating those changes to the integrator. Note that this functionality relies on translation since, like the data itself, changes to the data must be translated from the format and model of the information source into the format and model used by the warehousing system.

One approach is to ignore the change detection issue altogether and simply propagate entire copies of relevant data from the information source to the warehouse periodically. The integrator can combine this data with existing warehouse data from other sources, or it can request complete information from all sources and recompute the warehouse data from scratch. Ignoring change detection may be acceptable in certain scenarios, for example when it is not important for the warehouse data to be current and it is acceptable for the warehouse to be off-line occasionally. However, if currency, efficiency, and continuous access are required, then we believe that detecting and propagating changes and incrementally folding the changes into the warehouse will be the preferred solution.

In considering the change detection problem, we have identified several relevant types of information sources:

- **Cooperative sources:** Sources that provide triggers or other *active database* capabilities [19], so that notifications of changes of interest can be programmed to occur automatically.

- **Logged sources:** Sources maintaining a log that can be queried or inspected, so changes of interest can be extracted from the log.

- **Queryable sources:** Sources that allow the wrapper/monitor to query the information at the source, so that periodic polling can be used to detect changes of interest.

- **Snapshot sources:** Sources that do not provide triggers, logs, or queries. Instead, periodic dumps, or *snapshots*, of the data are provided off-line, and changes are detected by comparing successive snapshots.

Each type of information source capability provides interesting research problems for change detection. For example, in cooperative sources, although triggers and active databases have been explored in depth, putting such capabilities to use in the warehousing context still requires addressing the translation aspect; similarly for logged sources. In queryable sources, in addition to translation, one must consider performance and semantic issues associated with polling frequency: If the frequency is too high, performance will degrade, while if the frequency is too low, changes of interest may not be detected in a timely way. In snapshot sources, the challenge is to compare very large database dumps, detecting the changes of interest in an efficient and scalable way [13]. An important related problem in all of these scenarios is to develop appropriate representations for the changes to the data, especially if a non-relational model is used [4].

Finally, we note that a different wrapper/monitor component is needed for each information source, since the functionality of the wrapper/monitor is dependent on the type of the source (database system, legacy system, news wire, etc.) as well as on the data provided by that source. Clearly it is undesirable to hard-code a wrapper/monitor for each information source participating in a warehousing system, especially if new information sources become available frequently. Hence, a significant research issue is to develop techniques and tools that automate or semi-automate the process of implementing wrapper/monitors, through a toolkit or specification-based approach [16].

4.2 Integrator

Assume that the warehouse has been loaded with its initial set of data obtained from the information sources. (The task of setting up and loading the data warehouse is discussed in Section 4.5 below.) The ongoing job of the integrator is to receive change notifications from the wrapper/monitors for the information sources and reflect these changes in the data warehouse; see Figure 1.

At a sufficiently abstract level, the data in the warehouse can be seen as a *materialized view* (or set of views), where the *base data* resides at the information sources. Viewing the problem in this way, the job of the integrator is essentially to perform *materialized view maintenance* [9]. Indeed, there is a close connection between the view maintenance problem and data warehousing [15]. However, there are a number of reasons that conventional view maintenance techniques cannot be used, and each of these reasons highlights a research problem associated with data warehousing:

- In most data warehousing scenarios, the views stored at the warehouse tend to be more complicated than conventional views. For example, even if the warehouse and

[3] Most commercial data warehousing systems assume that both the information sources and the warehouse are relational, so translation is not an issue. However, some vendors do provide wrappers for other common types of information sources.

the information sources are relational, the views stored in the warehouse may not be expressible using a standard relational view definition language (such as SQL) over the base data. Typically, data warehouses may contain a significant amount of historical information (e.g., the history of stock prices or retail transactions), while the underlying sources may not maintain this information. Hence, warehouse views may not be functions of the underlying base data as traditional views are, but rather functions of the history of the underlying data. Relevant areas of research here certainly include temporal databases [18], as well as work on efficient monitoring of historical information [5].

• Data warehouses also tend to contain highly aggregated and summarized information [7]. Although in some cases aggregations may be describable in a conventional view definition language, the expressiveness of aggregates and summary operators in such languages are limited, so more expressive view definition languages may be needed. Furthermore, efficient view maintenance in the presence of aggregation and summary information appears to be an open problem [7, 17].

• The information sources updating the base data generally operate independently from the warehouse where the view is stored, and the base data may come from legacy systems that are unable or unwilling to participate in view maintenance. Most materialized view maintenance techniques rely on the fact that base data updates are closely tied to the view maintenance machinery, and view modification occurs within the same transaction as the updates. In the warehousing environment it is generally the case that:

- The system maintaining the view (the integrator) is only loosely coupled to the systems handling the base data (the information sources).
- The underlying information sources do not participate in view maintenance but simply report changes.
- Some sources may not provide locking capabilities, and there are almost certainly no global transactions.

In this scenario, certain "anomalies" arise when attempting to keep views consistent with base data (see [22]), and algorithms must be used that are considerably more complicated than conventional view maintenance algorithms.

• In a data warehouse, the views may not need to be refreshed after every modification or set of modifications to the base data. Rather, large batch updates to the base data may be considered, in which case efficient view maintenance techniques may involve different algorithms than are used for conventional view maintenance.

• In a data warehousing environment it may be necessary to transform the base data (sometimes referred to as *data scrubbing*) before it is integrated into the warehouse. Transformations might include, for example, aggregating or summarizing the data, sampling the data to reduce the size of the warehouse, discarding or correcting data suspected of being erroneous, inserting default values, or eliminating duplicates and inconsistencies.

Finally, we note that although integrators can be based purely on the data model used by the warehousing system, a different integrator still will be needed for each data warehouse, since a different set of views over different base data will be stored. As with wrapper/monitors, it is desirable not to require that each integrator be hard-coded from scratch, but rather to provide techniques and tools for generating integrators from high-level, nonprocedural specifications. This general approach is standard practice in conventional view maintenance, however there are a number of interesting problems in adapting it to data warehousing, discussed in the next section.

4.3 Warehouse Specification

In the previous section we drew an analogy between maintenance of a data warehouse and materialized view maintenance. We also indicated that it is useful to provide capabilities for specifying integrators in a high-level fashion, rather than implementing each integrator from scratch. Hence, in an ideal architecture, the contents of the data warehouse are specified as a set of view definitions, from which the warehouse updating tasks performed by the integrator and the change detection tasks required of the wrapper/monitors are deduced automatically.

For conventional view maintenance, algorithms have been developed to automatically generate active database rules for maintaining SQL-defined views [2]. Each rule is "triggered" by the notification of an update that may affect the view, and the rule modifies the view appropriately. A similar approach may be applied to data warehousing if a rule-driven integrator is used. Each integrator rule is triggered by a change notification (possibly of a specific type) from a wrapper/monitor. Similar to the view maintenance rules, integrator rules must update the warehouse to reflect the base data updates. However, in the warehousing scenario, rules may need to perform more complicated functions, such as fetching additional data from sources using remote queries [22] and "scrubbing" the data (as described in Section 4.2). Despite the additional complexity of rules in the warehousing environment, it still should be possible to automatically or semi-automatically generate appropriate rules from the warehouse (view) specification.

Thus, the research challenge in realizing the ideal architecture is to devise a warehouse specification language, rule capabilities, wrapper/monitor interfaces, and appropriate algorithms to permit developers of a data warehousing system to generate the integrator and the relevant change detection mechanisms automatically. We self-servingly note that this approach is being pursued by the *WHIPS* data warehousing project at Stanford [11].

4.4 Optimizations

In this section we outline three optimizations that can improve the performance of the architecture described Section 3: filtering irrelevant modifications at the sources, storing additional data at the warehouse for "self-maintainability," and efficiently managing multiple materialized views.

4.4.1 Update Filtering

We have said that all data modifications at a source that may be relevant to the warehouse are propagated to the integrator by the wrapper/monitor. Returning to our view maintenance analogy and considering the relational case as an example, we would propagate all inserts, deletes, and updates on any relation that participates in a view at the warehouse. A number of papers have been devoted to the topic of determining when certain modifications are guaranteed to leave a view unchanged, e.g., [14]. Related techniques allow distributed integrity constraints to be checked at a single site when certain types of modifications occur [10]. We believe that these classes of techniques can be adapted to data warehousing, whereby as many changes as possible are filtered at the source rather than propagated to the integrator.

4.4.2 Self-maintainability

When the integrator receives a change notification, in order to integrate that change into the warehouse the integrator may need to fetch additional data from the same or different sources. (As a simple example, if the warehouse joins two relations R and S, and there is a notification of an insert to relation R, then the inserted tuple must be joined with the contents of relation S.) Issuing queries to sources can incur a processing delay, the queries may be expensive, and such queries are the basis of the warehouse maintenance "anomalies" alluded to in Section 4.2 [22]. Even worse, when information sources are highly secure or when they are legacy systems, adhoc queries may not be permitted at all. Consequently, it may be desirable to ensure that, as much as possible, queries to the sources are not required in order to keep the warehouse data consistent.

In view maintenance, when additional queries over base data are never required to maintain a given view, then the view is said to be *self-maintainable* [1, 8]. Most views are not fully self-maintainable. However, self-maintainability can be ensured by storing additional data at the warehouse. For example, in the extreme case, all relevant data from the sources is copied to the warehouse, and views can be recomputed in their entirety if necessary. It appears to be an open research problem to determine the minimum amount of extra information needed for self-maintainability of a given view. Also interesting is to balance the cost of maintaining extra data at the warehouse against the cost of issuing queries to the sources.

4.4.3 Multiple View Optimization

Data warehouses may contain multiple views, for example to support different types of analysis. When these views are related to each other, e.g., if they are defined over overlapping portions of the base data, then it may be more efficient not to materialize all of the views, but rather to materialize certain shared "subviews," or portions of the base data, from which the warehouse views can be derived. When applicable, this approach can reduce storage costs at the warehouse and can reduce the effort required to integrate base data modifications into the warehouse. However, these savings must be balanced against slower query response at the warehouse, since some views may not be fully materialized.

4.5 Miscellaneous

We briefly note a few other important issues that arise in a data warehousing environment.

- **Warehouse management:** We have focused primarily on problems associated with the "steady state" of a data warehousing system. However, issues associated with warehouse design, loading, and metadata management are important as well. (In fact, it is these problems that have received the most attention from a large segment of the data warehousing industry to date.)

- **Source and warehouse evolution:** A warehousing architecture must gracefully handle changes to the information sources: schema changes, as well as the addition of new information sources and the removal of old ones. In addition, it is likely that clients will demand schema changes at the warehouse itself. All of these changes should be handled with as few disruptions or modifications to other components of the warehousing system as possible.

- **Duplicate and inconsistent information:** As in any environment involving multiple, heterogeneous information sources, there is the likelihood of encountering copies of the same information from multiple sources (represented in the same or different ways), or related information from multiple sources that is inconsistent. Earlier, we described the "scrubbing" of data from single sources. In addition, it is desirable for the integrator to scrub multi-source data, in order to eliminate duplicates and inconsistencies as much as possible.

- **Outdated information:** A feature of data warehouses is that they may contain historical information even when that information is not maintained in the sources. Nevertheless, in many cases it is undesirable to keep information "forever." Techniques are needed for specifying recency requirements in a warehousing environment, and for ensuring that outdated information is automatically and efficiently purged from the warehouse.

5 Conclusions

In the area of integrating multiple, distributed, heterogeneous information sources, data warehousing is a viable and in some cases superior alternative to traditional research solutions. Traditional approaches request, process, and merge information from sources when queries are posed. In the data warehousing approach, information is requested, processed, and merged continuously, so the information is readily available for direct querying and analysis at the warehouse.

Although the concept of data warehousing already is prominent in the database industry, we believe there are a number of important open research problems, described above, that need to be solved to realize the flexible, powerful, and efficient data warehousing systems of the future.

Acknowledgements

Thanks to Hector Garcia-Molina for recognizing the importance of data warehousing and introducing me to the topic, to the members of the *WHIPS* data warehousing project at Stanford (including Joachim Hammer, Wilburt Labio, Brian Lent, Dallan Quass, and Yue Zhuge), and to Elena Baralis, Stefano Ceri, Sudarshan Chawathe, Ashish Gupta, Venky Harinarayan, Yannis Papakonstantinou, and Dallan Quass for helpful comments on an initial draft of this paper.

References

[1] J.A. Blakeley, N. Coburn, and P.-A. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, September 1989.

[2] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 577–589, Barcelona, Spain, September 1991.

[3] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The Tsimmis project: Integration of heterogeneous information sources. In *Proceedings of 100th Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, October 1994.

[4] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. Technical report, Dept. of Computer Science, Stanford University, 1995. Available by anonymous ftp to `db.stanford.edu` in file `pub/chawathe/1995/tdiff2-0.ps`.

[5] J. Chomicki. History-less checking of dynamic integrity constraints. In *Proceedings of the Eighth International Conference on Data Engineering*, pages 557–564, Phoenix, Arizona, February 1992.

[6] IEEE Computer. *Special Issue on Heterogeneous Distributed Database Systems*, 24(12), December 1991.

[7] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational operator generalizing group-by, cross-tabs and sub-totals. *IEEE Transactions on Knowledge and Data Engineering*, 1995. To appear.

[8] A. Gupta, H.V. Jagadish, and I.S. Mumick. Data integration using self-maintainable views. Technical memorandum, AT&T Bell Laboratories, November 1994.

[9] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18, June 1995.

[10] A. Gupta and J. Widom. Local verification of global integrity constraints in distributed databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–58, Washington, D.C., May 1993.

[11] 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.

[12] W.H. Inmon and C. Kelley. *Rdb/VMS: Developing the Data Warehouse.* QED Publishing Group, Boston, Massachussetts, 1993.

[13] W. Labio and H. Garcia-Molina. Efficient snapshot differential algorithms in data warehousing. Technical report, Dept. of Computer Science, Stanford University, 1995. Available by anonymous ftp to `db.stanford.edu` in file `pub/labio/1995/window.ps`.

[14] A. Levy and Y. Sagiv. Queries independent of updates. In *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, pages 171–181, Dublin, Ireland, August 1993.

[15] D. Lomet and J. Widom, editors. *Special Issue on Materialized Views and Data Warehousing,* IEEE Data Engineering Bulletin 18(2), June 1995.

[16] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases*, Singapore, December 1995.

[17] R. Ramakrishnan, K.A. Ross, D. Srivastava, and S. Sudarshan. Efficient incremental evaluation of queries with aggregation. In *Proceedings of the International Logic Programming Symposium*, pages 204–218, 1994.

[18] M.D. Soo. Bibliography on temporal databases. *SIGMOD Record*, 20(1):14–24, March 1991.

[19] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann, San Francisco, California, 1995.

[20] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[21] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Data integration and warehousing using H20. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):29–40, June 1995.

[22] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 316–327, San Jose, California, May 1995.