# Issues in Developing Very Large Data Warehouses

Lyman Do Pamela Drew Wei Jin Vish Jumani David Van Rossum

Applied Research and Technology
Shared Services Group
The Boeing Company
P.O. Box 3707, M/S 7L-70, Seattle, WA 98124-2207, USA
Email: {Lyman.S.Do, Pamela.A.Drew, Wei.Jin, Vish.Jumani, David.A.VanRossum}@boeing.com

## Abstract

The size of The Boeing Company posts some stringent requirements on data warehouse design and implementation. We summarize four interesting and challenging issues in developing very large scale data warehouses, namely failure recovery, incremental update maintenance, cost model for schema design and query optimization, and metadata definition and management. For each issue, we give the reasons we think it is important but not well-addressed in research literature and commercial products, and our current research to solve it.

## 1 Introduction

Several data warehouse development projects are being pursued in Boeing with sizes ranging from hundreds of megabytes to terabytes. Some projects are aimed at providing sophisticated decision support and some are designed to re-distribute the workload of OLTP systems. In the latter, some large read-only queries will be re-directed to a data warehouse in order to relieve the heavy workload of our OLTP systems. This paper raises four issues that are challenging to the development of large-scale data warehouses.

The size of Boeing posts stringent requirements on data warehouse design. The largest data warehouse project will have an initial size of two to three terabytes and will operate on a 24x7 basis. Each airplane typically has over one million parts and the information of all parts and the airplane configuration will be kept in the warehouse until the airplane itself is retired. This means that the data lifecycle could be as long as 70+ years, instead of 5 to 10 years in most companies. This also implies that the data warehouse will continue to grow for 70 years. In addition to size, a long data lifecycle imposes a heavy requirement on the evolution of the data warehouse which has to be flexible enough to access 70 year old data. Moreover, since Boeing is a global company, there is virtually no "nighttime" for data warehouse refresh and maintenance. The refresh window is small and may not be extended. Any failure during refresh may miss the refresh window which delays business decisions or processes. The size of the data warehouse, the length of the data lifecycle, the flexibility to access data, and the strict demands on system availability establish requirements that challenge the most sophisticated technical solutions in data warehouse designs and implementations today.

In the following sections, we present four issues in turn and discuss the research directions that we are working toward.

## 2 Failure Recovery

Failure can happen anywhere, from populating the data warehouse, to refreshing multi-dimensional databases, to processing end-user queries; each task is long and resource intensive, and each task is costly to roll-back and restart. The situation becomes worse if the refresh window is small and can not be missed. This issue is not well addressed because failure recovery discussions typically only encompasses writing data into persistent storage but in a data warehouse environment, the most costly failures happen during computation such as data cleansing, aggregation/roll-

**Proceedings of the 24th VLDB Conference**
**New York, USA, 1998**

up, indexing, etc.

Typical transactional roll-back and restart failure recovery is not applicable in the above scenario because 1) it is too expensive (time and resources) to rollback a long-lived task, 2) the recovery log may not be available or too expensive to store due to the large volume of updates, and 3) in a global enterprise, the refresh window of a data warehouse is too small to allow rollback and re-start.

There are two types of failure that we have been looking into: (1) database/data warehouse update failure that happens during the data population and refresh from data sources to data warehouse and (2) computation failure during end-user query processing. The former is similar to a typical database failure except that it can happen even before writing records into the database during data cleansing and scrubbing, or it can happen during index or metadata updates.

The second type of failure happens during query processing. A typical decision support query may scan through multiple tables and require intensive computation to prepare summary information. Failure in any step during the computation requires re-work that is expensive in terms of query response time for mission critical queries, and computational resources. Also, re-work may eventually delay the warehouse refresh window. The influence of re-work and delay is further amplified since the data warehouse supports a large user population.

We are looking into the technology of incremental checkpointing to provide forward recovery. Ideally, all long-lived tasks, such as data cleansing, warehouse population and refresh, data summarization, indexing, roll-up, and query processing, should incrementally write checkpoints to a persistent storage. In case of failure, the system only needs to have partial roll-back to previous checkpoint and re-start. The concept is simple but technically challenging. Incremental checkpointing requires modularization of those long-lived tasks by analyzing and decomposing a long-lived task into a pipeline of sub-tasks each of which is loosely coupled with the others. Incremental checkpointing is performed between sub-tasks. Sub-tasks should be loosely coupled so that in case of failure, the system can roll-back to a previous checkpoint for each sub-task and re-start the pipeline. Another issue related to the incremental checkpointing is the need of an efficient and generic logging facility that provides persistent logging for checkpoints of different tasks.

## 3 Incremental Update Maintenance

In addition to the differential relation [OV91] approach of incremental update, we need a mechanism to support data sources that do not export differential relations. A differential relation captures the before image and the after image of all tuples that each operation affects. Most research work on data warehouse update focuses on the problem that "given a differential relation, how do we refresh the data warehouse efficiently." These works differentiate each other in terms of different data warehouse capabilities, such as convergent warehouse consistency [ZGHW95,ZHW96], replication of some source relations [QW97], versioning [QGMW96], etc. They are all based on the same assumption that differential relations are available. Likewise, commercial products either suggest refreshing the data warehouse from scratch (the snapshot approach) if the refresh window is large enough or support incremental update using differential relations.

Such an assumption may not be valid for the reason that some vital production systems do not export differential relations. Even if we violate the local autonomy by modifying the application code to extract differential relations from each database update, it is expensive to extract the before and after image of a SQL statement. To do this, the system needs to run a modified SQL statement with the same FROM and WHERE clauses as the original update operation before the execution of that update operation, then execute the update operation, then run the modified SQL statement again to collect the after image. Triggers could be helpful if the trigger can be fired before and after each execution (for INSERT, DELETE, and UPDATE) and if the triggers are tightly coupled with the update operation, i.e., being executed in a single atomic transaction. To further complicate the situation, some production systems use object wrappers to encapsulate relational schema or complete transactions and some commercial applications make it extremely difficult to interpret data storage structures, not to mention the feasibility of implementing triggers on them.

We have been working on incremental data warehouse update maintenance by capturing the operation descriptions at the sources. To differentiate, we refer to the differential relation as *value-delta* and the operation description as *operation-delta (Op-delta).* We are motivated by the fact that Op-delta could be extracted from database log and it is less expensive, in terms of storage, communication, and computation overhead, for programs to export Op-delta instead of the value-delta. For example, the statement: *"UPDATE status='revised' from parts where last_modified_date > 1/1/98"* may generate a value-delta in the size of ten thousand records but the SQL statement itself is already an Op-delta in the size of 70 bytes. We have identified sufficient conditions that Op-delta alone is enough to refresh the data warehouse (i.e., self-maintainability with respect to Op-delta), and for some cases, a hybrid between value-delta (the before image portion only) and the Op-delta is necessary to refresh the data warehouse. In both cases, the data warehouse does not need to refer back to the source during the refresh.

Another advantage of Op-delta is allowing the data warehouse to refresh concurrently with end-user query processing. The data warehouse treats a refresh as a series of execution of Op-delta. This implies that there will be virtually no downtime for data warehouse refresh, i.e., minimize or eventually eliminate the update window. To achieve this, we are currently working on the definition of data warehouse consistency in terms of concurrent refresh and concurrency protocol(s) for the data warehouse refresh.

## 4 Cost Model for Schema Design and Query Optimization

We need a cost model to analyze the cost and benefit of designing data warehouse schema. At large, the cost model should help in selection of data model among relational schema, star schema, and multi-dimensional database. At a finer granularity, the cost model should help in determining the dimensions of a multidimensional cube or in a star schema.

During the design of a data warehouse, an intuitive requirement is to maximize query performance. The resulting products are the star schema and multidimensional databases that pre-compute a sub-set of the most frequently asked queries (or asked by the most important person) and materialize the result. It is obvious that the query response time is tremendously improved but it is less obvious (or promoted) that a larger maintenance window is implied. Our traditional database training tells us that materialized views can improve query performance if we can manage to update the views consistently, i.e., we are trading data warehouse update maintenance cost for better query response time. But questions such as where is the balance point between improved query response time and the minimal maintenance window arise. There is no cost model to provide guidelines on how much information we should pre-compute and materialize, what kind of queries can benefit most from a materialized view, what is the cost to maintain a star schema or the equivalent multidimensional cube, etc.

In the Boeing Company, each airplane has potentially one million attributes to describe it. It is impossible to develop a multidimensional cube that has one million dimensions. An intuitive question will be "which attribute should we include in the limited multidimensional cube?" What is the benefit to introduce one more dimension and what will be the cost to maintain it? Again, we are looking for a cost model that can analyze the cost and benefit of bringing additional dimension into a multidimensional cube. The argument remains valid for a star schema. Assuming that the fact table is populated from tables in a normalized relation source, adding one more dimension table to the star schema means adding one more table to the join query that prepares the fact table. The cost to populate the fact table will then increase exponentially.

Last but not least, if a data warehouse cannot answer a query (determining whether a query is answerable is yet another issue), the query will then be reformulated and submitted to operational databases. What are the criteria that a query or a particular set of queries should be supported by the data warehouse? What is the cost of materializing additional relations/multidimensional cubes in the data warehouse in order to reduce the number of queries that have to be submitted to operational databases? Again, we need a cost model to analyze the balance among the cost of query processing at operational database, the cost of data warehouse update maintenance, and the benefit of supporting that query by the data warehouse.

## 5 Metadata Definition and Management

An orthogonal issue is the metadata definition and management. Example metadata includes: information about the data source such as the cost model of its query processing, whether value-delta is supported, whether it is possible to extract the Op-delta; information about the data such as source schema, data warehouse schema, and their mappings, update frequency and the average size of update; information about queries such as the cost to process a particular query at source, cost to process the same query if a multidimensional cube supports it, the frequency of the query, the importance (priority) of the query; information about the data warehouse such as the schema definition, subject area of each multidimensional cube/fact table, maintenance window, cost of maintenance, etc.

Metadata begins to accumulate at the very beginning of a data warehouse development project, either physically or electronically, and it grows during the development and beyond. We are particularly interested in the metadata that is 1) involved in the design/development decisions, 2) referred to during normal operations (data population, cleansing, refresh, etc.) of a data warehouse, and (3) referred to during end-user query processing. The first type of metadata includes the business model of data stored in operational databases and the cost model described in Section 4. This type of metadata is used to identify what information should be included in the data warehouse and from where to populate and refresh the identified information. The second type of metadata is used to maintain the data warehouse by identifying the methods to refresh the data warehouse, and for each part (multidimensional cubes, fact tables, etc) of the data warehouse, how to perform the refresh, how frequent the refresh should be performed, and when to archive the historical information. The third type of metadata helps users to identify information or subjects that are available in the data warehouse. It helps users to determine if a query is supported by the data warehouse. In an extreme, metadata helps the query processing system automatically route a query to the data warehouse or to the operational databases where the query can be executed, allowing a better response time and lower cost. End-user understanding of the data is generally not based on a relational type model, but a non-computing, business model, thus requiring a mapping of a business model (and possibly the business processes) to the physical data model used by the data warehouse. Due to the magnitude of the metadata, an online access with a business interpretation must be available.

After metadata is defined and the sources are identified, we need to manage changes to the metadata. Change management should capture changes at heterogeneous and distributed data sources and propagate the changes to a metadatabase. Besides, we need tools to analyze the changes and evolve the data warehouse. For example, changing the schema definition at a data source will change the metadata of that schema at the data warehouse and a mapping function between source to data warehouse tables/cubes. The change may also affect the self-maintainability of data warehouse tables/cubes, which in turn supports a more effective refresh mechanism. Likewise, changes of queries, their frequency, and priority may also trigger similar evolution at the data warehouse.

We are working on the definition of metadata, the schema definition in the "metadatabase", the software components in the "metadatabase" that supports queries on the metadata and that supports continuous update (change management) of the metadata.

## 6 Summary

Like most global enterprises, Boeing is looking into the data warehousing solutions to improve end-user query performance, to re-distribute some long-lived read-only queries from our overloaded OLTP systems, and to support a new-generation of decision support systems. We presented four interesting and challenging large-scale data warehouse development issues and we are actively working toward the solutions of them.

## References

Our references include the research work at AT&T Labs, Bell Labs, Stanford University, and various commercial data warehouse products. For brevity, we only include some of the references here. We direct interested readers to Alberto Mendelzon's "Data Warehousing and OLAP: A Research-Oriented Bibliography" web page at University of Toronto (http:\\ www.cs.toronto.edu/~mendel/dwbib.html).

[OV91] M.T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1991.

[QGMW96] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Proceedings of the Sixth International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, 1996.

[QW97] D. Quass and J. Widom. On-line warehouse view maintenance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 405-416, Tucson, Arizona, May 1997.

[ZGHW95] 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.

[ZGW96] Y. Zhuge, H. Garcia-Molina, and J.L. Wiener. The Strobe algorithms for multi-source warehouse consistency. In *Proceedings of Conference on Parallel and Distirbuted Information Systems*, Miami Beach, Florida, 1996.