JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 1, no. 4, September-October 2002

On Three Major Holes in Data Warehousing Today

Won Kim, Cyber Database Solutions, Inc. and MaxScan, Inc. , Austin, Texas

Abstract

During the past several years, many enterprises have created data warehouses (and data marts) in order to run applications. Data warehousing technologies used have been primarily extract-transform-load (ETL) tools, database modeling and design tools, and relational database (RDB) systems. There are three major holes in data warehousing today. These include inadequate attention to dirty data, inadequate performance and scalability in supporting scan-oriented operations, and inadequate selection of source data to load into the data warehouse. In this article, these three problematic areas are explored and approaches to addressing them are outlined.

1 INTRODUCTION

During the second half of the 1990s, many enterprises came to recognize that the data they had at their disposal is an important asset that, if properly leveraged, can give them competitive advantages. However, they realized that their data had been kept in disparate systems, and that in order to run applications, they had to integrate the data. This need to integrate enterprise data provided an impetus to the creation of data warehouses (or data marts, the "departmental equivalent" of data warehouses). Many enterprises invested heavily to create data warehouses and data marts, to run applications against them, and even re-engineer their business processes.

To create a data warehouse, an enterprise in general needs to iterate the following steps:

1. Analyze the data at its disposal to inventory the semantics and contents of the data in all relevant data sources.
2. Design (the database schema of) the data warehouse by considering the data available in all relevant data sources and the data needs of (i.e., queries to be generated by) the applications.
3. Extract appropriate data from the data sources, transform the extracted data to match the design of the data warehouse, and load the transformed data into the data warehouse.

Relational database systems (RDBs) are typically used to store and manage the data warehouses and data marts. Once a stable data warehouse has been created, it must be refreshed, as frequently as appropriate, to reflect updates to the data in the data sources.

Once a data warehouse has been created, an enterprise may run one or more applications against it. Applications typically include query tools for generating reports, customer relationship management (CRM) applications (e.g., marketing campaign tracking, customer segmentation, customer purchase behavior analysis), weblog data analyzers, data mining applications (e.g., fraud detection), business analytics (e.g., profitability analysis), etc. The applications generate SQL queries, which are passed to the RDBs that manage the data warehouse.

Despite the fairly large number of data warehouses and data marts that have been created and a fairly large number of applications that are being run, there are at least three major issues with data warehousing today. They include the disposition of dirty data, the optimal selection of source data, and performance and scalability against scan-oriented operations. The level of attention paid to the quality (correctness) of data and the impact (cost) of dirty data on the results of queries, data mining, and analytics is less than adequate. The issue of loading a data warehouse with all the data needed and only the data needed to answer queries generated by the applications has also not received adequate attention. RDBs perform well in evaluating queries that tend to select a small fraction of records from a large database, but come under the mercy of I/O speed when evaluating queries that require an entire table (or file) to be read from or written to disk. In this article, each of these issues will be explored and some approaches to addressing it will be discussed.

2 DATA QUALITY ISSUES

We encounter the results of dirty data in our daily lives. We receive mass mailing pieces with misspelled names, multiple mass mailing pieces with different aliases of the same name, mass mailing pieces addressed to people who moved out long ago, bank statements that show multiple withdrawals (for a single actual withdrawal), and so on. Dirty data includes missing data, incorrect data, and unusable data (e.g., due to being in a wrong format, not complying with the standard). Dirty data may arise for a variety of reasons, such as data entry errors, use of different representation formats or units of measurement, non-compliance with the standards, failure to update in a timely manner, failure to update all replicas of data, failure to remove duplicate records, and so on.

Obviously, the results of queries or data mining or business analytics against a data warehouse with a high proportion of dirty data cannot be reliable or usable. Only now enterprises are starting to adopt data cleansing tools to clean dirty data. Data cleansing tools on the market today, such as Vality/Ascential Software, Trillium Software, and First Logic, help to detect and automatically repair some important types of data, most notably names and addresses of people (using a nationwide directory of names and addresses). However, these tools still have a long way to go as they do not address all types of dirty data, and not enough enterprises make use of even such tools. Further, most enterprises do not employ credible methodologies and processes to ensure high-quality of data in their data warehouses. The reasons for the inadequate attention to data quality may include a lack of appreciation of the types and extent of dirty data that permeate data warehouses, the impact (cost) of dirty data on the business decisions made and actions taken, and also the fact that the data cleansing products on the market have not been well-enough marketed or are too pricey.

In order for the enterprises to start paying adequate attention to the quality of data in their data warehouses, they need first to come to understand the wide variety of dirty data that may exist, how they may come about, and how they may be prevented or detected and repaired. [Kim et al 2003] may be a reasonable starting point to this end. It presents a substantially comprehensive taxonomy of 33 types of dirty data, and also develops a corresponding taxonomy of techniques for preventing or detecting and repairing all types of dirty data. The taxonomy of dirty data reveals many types of dirty data that today's data-cleansing tools do not address. It also indicates various types of dirty data that can be automatically detected and repaired or even prevented. Frustratingly, however, it also exposes a large number of dirty data that do not appear to lend themselves readily to automated detection and repair or prevention. For example, the age of a person is erroneously entered as 26, when it is actually 25; or the name of a person is entered as "Larry Salcow", when it is actually "Larry Salchow"; or "Richard Guerrero at 4989 California Street, Thousand Oaks" and "R. Jorge Guerrero at 983 San Pedro Road, San Diego" refers to the old and current addresses of the same person. Data entry errors and failure to update the address of a person and failure to standardize the representation of a person's name are the causes of these dirty data. Clearly, it is practically impossible for any software (even a person) to detect that these are dirty data. Of course, in theory, extensive cross checks can be made against separate data sources (files or tables) that contain information on the same person's age, name, and address.

Next, enterprises need to come to appreciate that cost is associated with dirty data; that is, dirty data can actually bring about financial losses and legal liabilities if it is not prevented or detected and repaired. Suppose that a business sends out product promotional materials to 100,000 addresses at a cost of $2 per address. If 2% of the addresses were dirty such that the materials are undeliverable, 2,000 mailings would have been wasted at a cost of $4,000. (For simplicity, let us ignore the fact that actually 98% of typical mass mailings, even if correctly delivered, fall on deaf ears.)

The fact that dirty data has cost associated with it does not necessarily mean that it should be always prevented or detected and repaired. The reason is that it costs to prevent or detect or repair dirty data. The cost of dirty data must be weighed against the cost of preventing or detecting and repairing it. If it will take a 2-hour run of a data cleansing tool to correct the wrong addresses in the current example, and if the enterprise already has a data cleansing tool, most likely it makes sense to take the time to correct the addresses before mailing out the promotional materials. However, it will be better not to do anything about the wrong addresses if it will take paying a person $10,000 over a 100-

day period to manually check the accuracy of all addresses in a database against the telephone directory or a motor vehicle registration database.

Let us explore the cost of preventing or detecting and repairing dirty data. Any type of dirty data may be prevented or detected and repaired, automatically or manually, but at a cost. In general, there is more than one way to prevent or detect and repair any given type of dirty data, each at a different cost. For example, the transaction management facilities in RDBs prevent certain types of dirty data, such as lost updates, dirty reads, and non-repeatable reads [Kim et al 2003]. Such integrity constraints as data types, Unique, Null Not Allowed, and foreign key, once specified, cause RDBs to automatically enforce integrity of data against inserts, updates, and deletes. These facilities are part of RDBs and take up only a relatively small amount of computer time. A spell checker may be used to detect some misspelled words. A directory of names and addresses may be used to repair misspelled names and addresses, and fill in additional address information, such as the zip code, county name, etc. These typically require human intervention to accept the recommendations of software tools. For certain types of dirty data, automated detection is possible to a good extent, but absolute correctness may be impossible to guarantee. For example, an automatic value-range checker may be used to ensure that a person's age is within a range, say 18 and 67. To ensure that there was no data entry error, more than one data entry persons may be assigned to check and double-check each data entry.

The cost of repairing dirty data also depends on the volume of data involved and the proportion of dirty data. Obviously, a file with a larger number of records and a larger number of fields will take a greater effort to check than a file with a smaller number of records and a smaller number of fields. The cost of detecting and repairing a single dirty data in a single field in a single record also varies depending on the type of dirty data.

Most enterprises today certainly do not do enough to ensure high quality of data in their data warehouses. To ensure high quality of data, enterprises need to have a process, methodologies and resources to monitor and analyze the quality of data, methodologies for preventing and/or detecting and repairing dirty data, and methodologies for measuring the cost of dirty data and the cost of ensuring high quality of data. DAQUM (Data Quality Measurement) is a prototype tool developed at Ewha Women's University for monitoring most types of dirty data and assigning quantitative measures of data quality to different types of dirty data under different application situations [Kim et al 2002]. Additional efforts along this line are needed.

3 SOURCE DATA SELECTION ISSUES

Today data warehouse designers design the database schema of the target data warehouse using a database-modeling tool. The database schema consists of tables, columns (fields) within the tables, data types for the columns and constraints on the columns, and relationships between tables. The designers also specify the mappings (transformations) from the schemas of the data sources to the schema of the target data warehouse.

But how do the designers determine that the data warehouse contains all the data required by the applications that will be run against it, and that the data warehouse does not contain any data not needed by the applications? Today, it is an educated guesswork by experienced designers. The designers have to elicit the data needs (tables and columns) by interviewing application developers, business analysts (people who understand the data needs of applications and businesses), and database administrators. When a data warehouse is initially created, often it is missing certain data needed to answer certain types of queries, and also has data that applications do not ever need. Although storage may be relatively cheap, all fields, both needed and unneeded, are stored in the same records and are stored and retrieved together, slowing down retrieval time, adding to the processing time, as well as taking up storage space needlessly.

In the research literature, there are many proposals for modeling the data warehouse as a repository of the results of all the queries that are run [Gupta 1997][Yang et al 1997][Kotidis and Roussopoulos 1999]. These proposals attempt to find algorithms that will select (for loading into the data warehouse) a subset of the source data that will minimize the total query response time. Some attempt to also minimize the cost of updating the data warehouse. In other words, they start with the assumption that all queries against a data warehouse can be obtained or predicted, and that all updates against the data warehouses, and thus the original data sources, can be obtained or predicted.

An ideal way to select data to load into a data warehouse is to first determine all the queries that will be generated by all the applications to run against the data warehouse, and determine all the tables and fields included in the queries. Determining all the queries in advance of creating a data warehouse is difficult. However, after the initial creation of a data warehouse, by logging all the queries generated by all the applications for a reasonable period of time, it may become possible. An analysis of the queries logged may be used to fine-tune the data warehouse, by removing data that is not accessed by the applications.

A potentially helpful and practical tool is one that can take the applications' data needs, automatically match them against the schemas of the data sources, and recommend an optimal subset of the data sources that need to be loaded into the data warehouse, such that all the data needed are in the data warehouse and only the data needed are in the data warehouse. MaxCentra is such a tool that has become commercially available recently [Kim et al 2000]. The only dependence MaxCentra has is a pre-built knowledgebase of keywords that represent the data needs of the applications. The keywords are basically hints for the tables and fields to be accessed or retrieved in queries generated by the applications. Such a list of keywords must be provided by business analysts or application developers, or may be automatically collected from the queries logged from the applications that run against a non-optimal data warehouse. MaxCentra starts from there and, with help and confirmation by the data warehouse designer, arrives at an optimal database schema for the data warehouse. MaxCentra includes several computational stages, and the data warehouse designer may confirm or modify the results at the end of each stage. If MaxCentra is only given keywords for business hints, rather than logged set of queries, it performs all standard processing of the keywords (stemming of words, decomposition of compound words, similar words, etc.). Then it rank orders the tables and fields in the data sources in terms of relevance to the data needs of the applications, groups the tables and fields that may be redundant or may be derived from one another (so that redundant or non-essential fields or tables may be removed), and rank orders the groups in terms of relevance to the data needs of the applications.

4 PERFORMANCE AND SCALABILITY ISSUES

RDBs use access methods such as hashing and/or B+-tree indexes to retrieve a small number of desired records without having to scan the entire table or database. Such access methods are highly effective in answering queries that are keyed on a single field (or a small number of fields) when the results are small fractions of the entire table. Examples of such queries are “find all 25-year olds”, “find all software engineers”, and “find all 25-year old software engineers”. To answer such queries fast, an index may be created on the “Age” column in the “Employee” table, the “Job” column in the “Employee” table, and the “Age” and “Job” columns jointly in the “Employee” table, respectively.

However, access methods are in general not helpful in answering queries whose results are substantial parts of a table. Examples are “find all females”, “find all non-smokers”, and “find young employees”. Further, access methods are not helpful if the values in the column are to undergo frequent updates, as such updates will require the access methods to be recreated. These are examples of even “simple” queries that render access methods in RDBs useless.

Besides these “simple” queries, there are two classes of operations that render access methods in RDBs powerless. One is the “aggregation” operation, including the grouping of all records in a table and computing aggregation functions on the grouped records (average, total, sum, min, and max). This type of operation is important in such applications as weblog data analysis, customer data segmentation, and so on. Various techniques for addressing the performance issues for this class of operations are explored in [Kim 1998], including data compression, field-based table storage technique (storing each field of a table as a separate file), pre-calculation (of OLAP cubes, summary tables) from detailed data, normalization (partitioning of a table into multiple smaller tables), and parallel-processing. MaxScan and Ab Initio are software products on the market that are designed to address the performance and scalability issues for this type of operation. MaxScan employs a field-based table-storage technique, hashing techniques for grouping and aggregating records, and parallel processing techniques. On average it achieves a 10-20 times performance and scalability improvement over RDBs for the aggregation operation. Ab Initio is an ETL tool in which the data transformation engine employs performance techniques.

Another is the “file movement” operation, that is, operations that read entire file(s) and/or write out entire file(s). This type of operation is important in the time-consuming "data transformation" phase in creating a data warehouse or the "data preparation" phase in automatic knowledge discovery (data mining) from stored data [Pyle 1999]. The data transformation phase involves the transformation of the format and representation of data in a given field (unit change, date time format change, abbreviation changes, etc.), merging two or more fields into one, splitting a field into two or more fields, sorting a table, creating a summary table from a table with detailed data, creating a new table by joining two or more tables, merging two or more tables into a single table, splitting a table into two or more tables, and so on. The data preparation phase involves the transformation of data in a given field into numeric code (for neural networks), the transformation of continuous numerical data in a given field into categorical data (e.g., 60 and higher in age as 'old'), adding a new field to a record, taking samples of data from a table, replicating certain records in a table (to achieve a desired distribution of records), and so on. More detailed discussions of these operations are given in [Kim 1999].

Today these file-movement operations are almost entirely at the mercy of the sequential file operations of RDBs, that is, the reading of one or more files, the creating of temporary files, and the writing of resulting file or files. The frequency of such operations and the volume of data involved may justify a dedicated data transformation/preparation server. Such a server may ideally consist of multiple CPUs running in parallel. Regardless of the CPU configuration, it should run data transformation/preparation software tools that are designed for parallel processing. Whenever the workflow justifies it, pipelined parallel processing needs to be employed, where incremental results of one operation feed the next operation without having to wait for the completion of the first operation. Pipelined parallel processing obviates the need to have the full results of one operation written into a temporary file and read by the subsequent operation, saving two file I/Os. To create summary tables, it may make sense to use a fast sort engine, such as SyncSort, or a fast aggregation engine, such as MaxScan, rather than using the native functions of RDBs. Further, to perform record-by-record computations (sampling, transformation of data format or representation, etc.), it may make sense to partition a file into multiple sub-files and assign them to different CPUs for parallel processing.

5 CONCLUDING REMARKS

In this article we identified three major issues that have not been adequately addressed in data warehousing: data quality, optimal source data selection, and performance and scalability. There are a few data cleansing tools on the market and they are starting to be used to cleanse various types of dirty data. However, these tools certainly do not address all types of dirty data, and certainly not many enterprises have adopted either the tools or the process for preventing or detecting and repairing dirty data and for monitoring and quantifying data quality in their data warehouses. Today, data warehouses contain lots of data that are never needed by the applications that run against them, and such unneeded data is one of the sources that slow down query performance. It should be possible to log a complete set of queries that are generated by all applications, and use the tables and fields included in the queries to fine-tune the contents of the data warehouses. Today's data warehouses largely use RDBs for storing and managing data. However, RDBs today are not adequate in processing scan-oriented queries such as grouping records and computing aggregations, and file-movement operations that predominate in the data transformation phase of data warehousing or the data preparation phase of data mining.

REFERENCES

[Gupta 1997] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In Proceedings of the Intl. Conf. on Database Theory, pages 98-112, January 1997.

[Kim 1998] W. Kim. "Business Intelligence at Top Speed", Intelligent Enterprise, Miller Freeman, Inc., Dec. 15, 1998.

[Kim 1999] W. Kim. "I/O Problems in Preparing Data for Data Warehousing and Data Mining – Part 1", Journal of Object-Oriented Programming, 101 Communications, Feb. 1999.

[Kim et al 2000] W. Kim, E. Hong, K. Kim, D. Lee. "A Component-Based Architecture for Preparing Data in Data Warehousing", Journal of Object-Oriented Programming, March/April 2000.

[Kim et al 2002] W. Kim, et al. "The Chamois Component-Based Knowledge Engineering Framework," IEEE Computer, May 2002, pp. 44-52, IEEE CS Press.

[Kim et al 2003] W. Kim, B. Choi, E. Hong, S. Kim, D. Lee. "A Taxonomy of Dirty Data", *Journal of Data Mining and Knowledge Discovery*, the Kluwer Academic-Publishers, to appear in 2003

[Kotidis and Roussopoulos 1999] Yannis Kotidis and Nick Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. In Proc. of ACM SIGMOD Conference, pages 371-382, June 1999

[Pyle 1999] D. Pyle. Data Preparation for Data Mining, Morgan Kaufmann Publishers, 1999.

[Yang et al 1997] J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In Proc. of the 23th VLDB Conference, pages 136-145, August 1997.

About the author

Won Kim is President and CEO of Cyber Database Solutions (www.cyberdb.com) and MaxScan (www.maxscan.com) in Austin, Texas, USA. He is also Dean of Ewha Institute of Science and Technology, Ewha Women's University, Seoul. Korea. He is Editor-in-Chief of ACM Transactions on Internet Technology (www.acm.org/toit), and Chair of ACM Special Interest Group on Knowledge Discovery and Data Mining (www.acm.org/sigkdd). He is the recipient of the ACM 2001 Distinguished Service Award.