© 2014 IJIRT | Volume 1 Issue 6 | ISSN : 2349-6002

An Overview of Data Warehousing and OLAP Technology

Sweta Singh, Parul Malhan
Student, B.Tech, Department of Electronics and Computers Engineering
Dronacharya College of Engineering, Gurgaon, India

***Abstract*- Data warehousing and on-line analytical processing (OLAP) are essential elements of decision support, which has increasingly become a focus of the database industry. Many commercial products and services are now available, and all of the principal database management system vendors now have offerings in these areas. Decision support places some rather different requirements on database technology compared to traditional on-line transaction processing applications. This paper provides an overview of data warehousing and OLAP technologies, with an emphasis on their new requirements. We describe back end tools for extracting, cleaning and loading data into a data warehouse; multidimensional data models typical of OLAP; front end client tools for querying and data analysis; server extensions for efficient query processing; and tools for metadata management and for managing the warehouse. In addition to surveying the state of the art, this paper also identifies some promising research issues, some of which are related to problems that the database research community has worked on for years, but others are only just beginning to be addressed. This overview is based on a tutorial that the authors presented at the VLDB Conference, 1996.**

***Index Terms*- Data warehouse, OLAP, architecture, utility tools, Design**

I. INTRODUCTION

Data warehousing is a collection of *decision support* technologies, aimed at enabling the *knowledge worker* (executive, manager, and analyst) to make better and faster decisions. The past three years have seen explosive growth, both in the number of products and services offered and in the adoption of these technologies by industry. According to the *META Group*, the data warehousing market, including hardware, database software, and tools, is projected to grow from $2 billion in 1995 to $8 billion in 1998. Data warehousing technologies have been successfully deployed in many industries: manufacturing (for order shipment and customer support), retail (for user profiling and inventory management), financial services (for claims analysis, risk analysis, credit card analysis, and fraud detection), transportation (for fleet management), telecommunications (for call analysis and fraud detection), utilities (for power usage analysis), and healthcare (for outcomes analysis). This paper presents a roadmap of data warehousing technologies, focusing on the special requirements that data warehouses place on database management systems (DBMSs).

A data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making."1 Typically, the data warehouse is maintained separately from the organization's operational databases. There are many reasons for doing this. The data warehouse supports on-line analytical processing (OLAP), the functional and performance requirements of which are quite different from those of the on-line transaction processing (OLTP) applications traditionally supported by the operational databases.

OLTP applications typically automate clerical data processing tasks such as order entry and banking transactions that are the bread-and-butter day-to-day operations of an organization. These tasks are structured and repetitive, and consist of short, atomic, isolated transactions. The transactions require detailed, up-to-date data, and read or update a few (tens of) records accessed typically on their primary keys. Operational databases tend to be hundreds of megabytes to gigabytes in size. Consistency and recoverability of the database are critical, and maximizing transaction throughput is the key performance metric. Consequently, the database is designed to reflect the operational semantics of known applications, and in particular, to minimize concurrency conflicts.

Data warehouses, in contrast, are targeted for decision support. Historical, summarized and consolidated data

is more important than detailed, individual records. Since data warehouses contain consolidated data, perhaps from several operational databases, over potentially long periods of time, they tend to be orders of magnitude larger than operational databases; enterprise data warehouses are projected to be hundreds of gigabytes to terabytes in size. The workloads are query intensive with mostly ad hoc, complex queries that can access millions of records and perform a lot of scans, joins, and aggregates. Query throughput and response times are more important than transaction throughput.

To facilitate complex analyses and visualization, the data in a warehouse is typically modeled *multi-dimensionally*. For example, in a sales data warehouse, time of sale, sales district, Sales person, and product might be some of the dimensions of interest. Often, these dimensions are hierarchical; time of sale may be organized as a day-month-quarter-year hierarchy, product as a product-category-industry hierarchy. Typical OLAP operations include *rollup* (increasing the level of aggregation) and *drill-down* (decreasing the level of aggregation or increasing detail) along one or more dimension hierarchies, *slice_and_dice* (selection and projection), and *pivot* (re-orienting the multidimensional view of data).

Given that operational databases are finely tuned to support known OLTP workloads, trying to execute complex OLAP queries against the operational databases would result in unacceptable performance. Furthermore, decision support requires data that might be missing from the operational databases; for instance, understanding trends or making predictions requires historical data, whereas operational databases store only current data. Decision support usually requires consolidating data from many heterogeneous sources: these might include external sources such as stock market feeds, in addition to several operational databases. The different sources might contain data of varying quality, or use inconsistent representations, codes and formats, which have to be reconciled. Finally, supporting the multidimensional data models and operations typical of OLAP requires special data organization, access methods, and implementation methods, not generally provided by commercial DBMSs targeted for OLTP. It is for all these reasons that data warehouses are implemented separately from operational databases.

Data warehouses might be implemented on standard or extended relational DBMSs, called Relational OLAP (ROLAP) servers. These servers assume that data is stored in relational databases, and they support extensions to SQL and special access and implementation methods to efficiently implement the multidimensional data model and operations. In contrast, multidimensional OLAP (MOLAP) servers are servers that directly store multidimensional data in special data structures (e.g., arrays) and implement the OLAP operations over these special data structures.

Research in data warehousing is fairly recent, and has focused primarily on query processing and view maintenance issues. There still are many open research problems. Ongoing research efforts are concentrated on creating a benchmark for combined OLTP and OLAP systems, which is derived from real customer systems and data, multi-tenancy for in-memory column data bases as well as optimizations around the delta merge process.

II. ARCHITECTURE AND END-TO-END PROCESS

Figure 1 shows a typical data warehousing architecture.

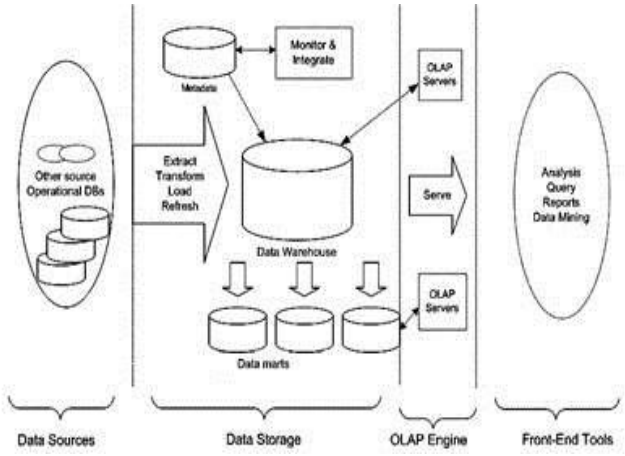

Figure 1. Data Warehousing Architecture

It includes tools for extracting data from multiple operational databases and external sources; for cleaning, transforming and integrating this data; for loading data into the data warehouse; and for periodically refreshing the warehouse to reflect updates at the sources and to purge data from the warehouse, perhaps onto slower archival storage. In addition to the main warehouse, there may be several departmental data marts. Data in the warehouse and data marts is stored and managed by one or more warehouse servers, which present multidimensional views of data to a

variety of front end tools: query tools, report writers, analysis tools, and data mining tools. Finally, there is a repository for storing and managing metadata, and tools for monitoring and administering the warehousing system.

The warehouse may be distributed for load balancing, scalability, and higher availability. In such a distributed architecture, the metadata repository is usually replicated with each fragment of the warehouse, and the entire warehouse is administered centrally. An alternative architecture, implemented for expediency when it may be too expensive to construct a single logically integrated enterprise warehouse, is a federation of warehouses or data marts, each with its own repository and decentralized administration.

Designing and rolling out a data warehouse is a complex process, consisting of the following activities:

- Define the architecture, do capacity planning, and select the storage servers, database and OLAP servers, and tools.
- Integrate the servers, storage, and client tools.
- Design the warehouse schema and views.
- Define the physical warehouse organization, data placement, partitioning, and access methods.
- Connect the sources using gateways, ODBC drivers, or other wrappers.
- Design and implement scripts for data extraction, cleaning, transformation, load, and refresh.
- Populate the repository with the schema and view.
- Definitions, scripts, and other metadata.
- Design and implement end-user applications.
- Roll out the warehouse and applications.

III. BACK END TOOLS AND UTILITIES

Data warehousing systems use a variety of data extraction and cleaning tools, and load and refresh utilities for populating warehouses. Data extraction from "foreign" sources is usually implemented via gateways and standard interfaces (such as Information Builders EDA/SQL, ODBC, Oracle Open Connect, Sybase Enterprise Connect, and Informix Enterprise Gateway).

Data Cleaning

Since a data warehouse is used for decision making, it is important that the data in the warehouse be correct. However, since large volumes of data from multiple sources are involved, there is a high probability of errors and anomalies in the data.. Therefore, tools that help to detect data anomalies and correct them can have a high payoff. Some examples where data cleaning becomes necessary are: inconsistent field lengths, inconsistent descriptions, inconsistent value assignments, missing entries and violation of integrity constraints. Not surprisingly, optional fields in data entry forms are significant sources of inconsistent data.

There are three related, but somewhat different, classes of data cleaning tools. *Data migration* tools allow simple transformation rules to be specified; e.g., "replace the string *gender* by *sex*". Warehouse Manager from Prism is an example of a popular tool of this kind. *Data scrubbing* tools use domain-specific knowledge (e.g., postal addresses) to do the scrubbing of data. They often exploit parsing and fuzzy matching techniques to accomplish cleaning from multiple sources. Some tools make it possible to specify the "relative cleanliness" of sources. Tools such as Integrity and Trillium fall in this category. *Data auditing* tools make it possible to discover rules and relationships (or to signal violation of stated rules) by scanning data. Thus, such tools may be considered variants of data mining tools. For example, such a tool may discover a suspicious pattern (based on statistical analysis) that a certain car dealer has never received any complaints.

Load

After extracting, cleaning and transforming, data must be loaded into the warehouse. Additional preprocessing may still be required: checking integrity constraints; sorting; summarization, aggregation and other computation to build the derived tables stored in the warehouse; building indices and other access paths; and partitioning to multiple target storage areas. Typically, batch load utilities are used for this purpose. In addition to populating the warehouse, a load utility must allow the system administrator to monitor status, to cancel, suspend and resume a load, and to restart after failure with no loss of data integrity.

The load utilities for data warehouses have to deal with much larger data volumes than for operational databases. There is only a small time window (usually at night) when the warehouse can be taken offline to refresh it. Sequential loads can take a very long time, e.g., loading a terabyte of data can take weeks and months! Hence, pipelined and partitioned parallelism is typically exploited. Doing a full load has the advantage

that it can be treated as a long batch transaction that builds up a new database. While it is in progress, the current database can still support queries; when the load transaction commits, the current database is replaced with the new one. Using periodic checkpoints ensures that if a failure occurs during the load, the process can restart from the last checkpoint.

However, even using parallelism, a full load may still take too long. Most commercial utilities (e.g., Red Brick Table
Management Utility) use incremental loading during refresh to reduce the volume of data that has to be incorporated into the warehouse. Only the updated tuples are inserted. However, the load process now is harder to manage. The incremental load conflicts with ongoing queries, so it is treated as a sequence of shorter transactions (which commit periodically, e.g., after every 1000 records or every few seconds), but now this sequence of transactions has to be coordinated to ensure consistency of derived data and indices with the base data.

Refresh

Refreshing a warehouse consists in propagating updates on source data to correspondingly update the base data and derived data stored in the warehouse. There are two sets of issues to consider: *when* to refresh, and *how* to refresh. Usually, the warehouse is refreshed periodically (e.g., daily or weekly). Only if some OLAP queries need current data (e.g., up to the minute stock quotes), is it necessary to propagate every update. The refresh policy is set by the warehouse administrator, depending on user needs and traffic, and may be different for different sources.

Refresh techniques may also depend on the characteristics of the source and the capabilities of the database servers.
Extracting an entire source file or database is usually too expensive, but may be the only choice for legacy data sources. Most contemporary database systems provide replication servers that support incremental techniques for propagating updates from a primary database to one or more replicas. Such replication servers can be used to incrementally refresh a warehouse when the sources change. There are two basic replication techniques: data shipping and transaction shipping.

In data shipping (e.g., used in the Oracle Replication Server, Praxis Omni Replicator), a table in the warehouse is treated as a remote snapshot of a table in the source database. *After row* triggers are used to update a snapshot log table whenever the source table changes; and an automatic refresh schedule (or a manual refresh procedure) is then set up to propagate the updated data to the remote snapshot.

In transaction shipping (e.g., used in the Sybase Replication Server and Microsoft SQL Server), the regular transaction log is used, instead of triggers and a special snapshot log table. At the source site, the transaction log is sniffed to detect updates on replicated tables, and those log records are transferred to a replication server, which packages up the corresponding transactions to update the replicas. Transaction shipping has the advantage that it does not require triggers, which can increase the workload on the operational source databases. However, it cannot always be used easily across DBMSs from different vendors, because there are no standard APIs for accessing the transaction log.

Such replication servers have been used for refreshing data warehouses. However, the refresh cycles have to be properly chosen so that the volume of data does not overwhelm the incremental load utility.

In addition to propagating changes to the base data in the warehouse, the derived data also has to be updated correspondingly. The problem of constructing logically correct updates for incrementally updating derived data (materialized views) has been the subject of much research. For data warehousing, the most significant classes of derived data are summary tables, single-table indices and join indices.

IV. CONCEPTUAL MODEL AND FRONT END TOOLS

A popular conceptual model that influences the front-end tools, database design, and the query engines for OLAP is the *multidimensional* view of data in the warehouse. In a multidimensional data model, there is a set of *numeric measures* that are the objects of analysis. Examples of such measures are sales, budget, revenue, inventory, ROI (return on investment). Each of the numeric measures depends on a set of *dimensions*, which provide the context for the measure.

For example, the dimensions associated with a sale amount can be the city, product name, and the date

when the sale was made. The dimensions together are assumed to *uniquely* determine the measure. Thus, the multidimensional data views a measure as a value in the multidimensional space of dimensions. Each dimension is described by a set of attributes. For example, the Product dimension may consist of four attributes: the category and the industry of the product, year of its introduction, and the average profit margin. For example, the soda Surge belongs to the category beverage and the food industry, was introduced in 1996, and may have an average profit margin of 80%. The attributes of a dimension may be related via a hierarchy of relationships. In the above example, the product name is related to its category and the industry attribute through such a hierarchical relationship.

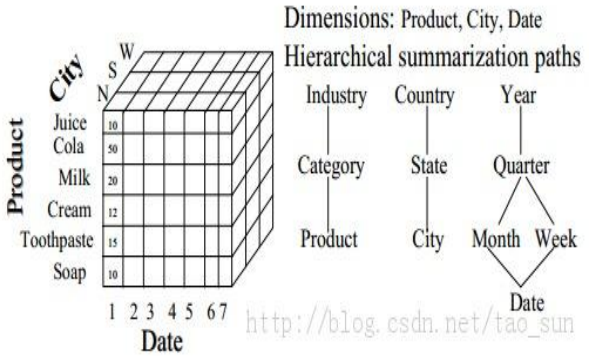

Figure 2. Multidimensional data model

Another distinctive feature of the conceptual model for OLAP is its stress on *aggregation* of measures by one or more dimensions as one of the key operations; e.g., computing and ranking the *total* sales by each county (or by each year). Other popular operations include *comparing* two measures (e.g., sales and budget) aggregated by the same dimensions. Time is a dimension that is of particular significance to decision support (e.g., trend analysis). Often, it is desirable to have built-in knowledge of calendars and other aspects of the time dimension.

Front End Tools

The multidimensional data model grew out of the view of business data popularized by PC spreadsheet programs that were extensively used by business analysts. The spreadsheet is still the most compelling front-end application for OLAP. The challenge in supporting a query environment for OLAP can be crudely summarized as that of supporting spreadsheet operations efficiently over large multi-gigabyte databases.

We shall briefly discuss some of the popular operations that are supported by the multidimensional spreadsheet applications. One such operation is *pivoting*. Consider the multidimensional schema of Figure 2 represented in a spreadsheet where each row corresponds to a sale. Let there be one column for each dimension and an extra column that represents the amount of sale. The simplest view of pivoting is that it selects two dimensions that are used to aggregate a measure, e.g., sales in the above example. The aggregated values are often displayed in a grid where each value in the (x, y) coordinate corresponds to the aggregated value of the measure when the first dimension has the value x and the second dimension has the value y. Thus, in our example, if the selected dimensions are city and year, then the x-axis may represent all values of city and the y-axis may represent the years. The point (x, y) will represent the aggregated sales for city x in the year y. Thus, what were values in the original spreadsheets have now become row and column headers in the pivoted spreadsheet.

Other operators related to pivoting are *rollup* or *drill-down.* Rollup corresponds to taking the current data object and doing a further group-by on one of the dimensions. Thus, it is possible to roll-up the sales data, perhaps already aggregated on city, additionally by product. The drill-down operation is the converse of rollup. *Slice_and_dice* corresponds to reducing the dimensionality of the data, i.e., taking a projection of the data on a subset of dimensions for selected values of the other dimensions. For example, we can slice_and_dice sales data for a specific product to create a table that consists of the dimensions city and the day of sale. The other popular operators include *ranking* (sorting), *selections* and defining *computed* attributes.

Although the multidimensional spreadsheet has attracted a lot of interest since it empowers the end user to analyze business data, this has not replaced traditional analysis by means of a *managed query environment.* These environments use stored procedures and predefined complex queries to provide packaged analysis tools. Such tools often make it possible for the end-user to query in terms of domain-specific business data. These applications often use raw data access tools and optimize the access patterns depending on the back end database server. In addition, there are query environments (e.g., Microsoft Access) that help build *ad hoc* SQL queries by "pointing-and-

clicking". Finally, there are a variety of data mining tools that are often used as front end tools to data warehouses.

V. DATABASE DESIGN METHODOLOGY

The multidimensional data model described above is implemented directly by MOLAP servers. We will describe these briefly in the next section. However, when a relational ROLAP server is used, the multidimensional model and its operations have to be mapped into relations and SQL queries. In this section, we describe the design of relational database schemas that reflect the multidimensional views of data.

Entity Relationship diagrams and normalization techniques are popularly used for database design in OLTP environments. However, the database designs recommended by ER diagrams are inappropriate for decision support systems where efficiency in querying and in loading data (including incremental loads) are important.

Most data warehouses use a *star schema* to represent the multidimensional data model. The database consists of a single fact table and a single table for each dimension. Each tuple in the fact table consists of a pointer (foreign key – often uses a generated key for efficiency) to each of the dimensions that provide its multidimensional coordinates, and stores the numeric measures for those coordinates. Each dimension table consists of columns that correspond to attributes of the dimension. Figure 3 shows an example of a star schema.

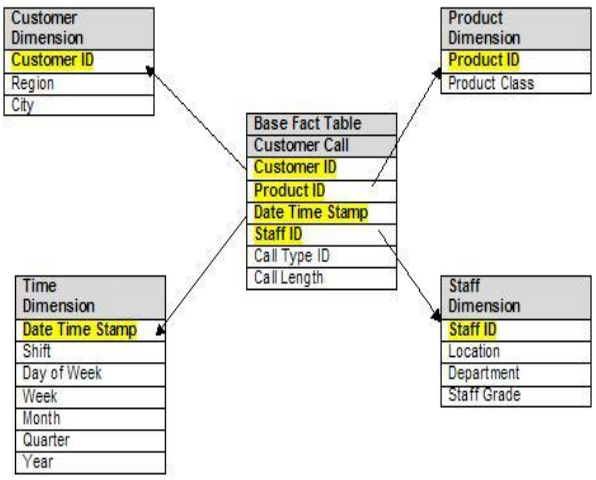

Figure 3. A Star Schema

Star schemas do not explicitly provide support for attribute hierarchies. *Snowflake schemas* provide a refinement of star schemas where the dimensional hierarchy is explicitly represented by normalizing the dimension tables, as shown in Figure 4. This leads to advantages in maintaining the dimension tables. However, the de-normalized structure of the dimensional tables in star schemas may be more appropriate for browsing the dimensions.

Fact constellations are examples of more complex structures in which multiple fact tables share dimensional tables. For example, projected expense and the actual expense may form a fact constellation since they share many dimensions.

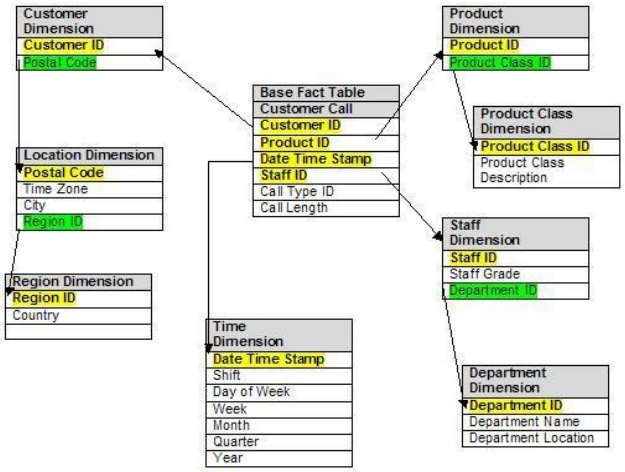

Figure 4. A Snowflake Schema

In addition to the fact and dimension tables, data warehouses store selected summary tables containing pre-aggregated data. In the simplest cases, the pre-aggregated data corresponds to aggregating the fact table on one or more selected dimensions. Such pre-aggregated summary data can be represented in the database in at least two ways. Let us consider the example of a summary table that has total sales by product by year in the context of the star schema of Figure 3. We can represent such a summary table by *a separate fact table* which shares the dimension Product and also *a separate shrunken dimension table* for time, which consists of only the attributes of the dimension that makes sense for the summary table (i.e., year). Alternatively, we can represent the summary table by encoding the aggregated tuples in the *same fact table* and the *same dimension tables* without adding new tables.

This may be accomplished by adding a new *level* field to each dimension and using *nulls*: We can encode a day, a month or a year in the Date dimension table as follows: (id0, 0, 22, 01, 1960) represents a record for Jan 22, 1960, (id1, 1, NULL, 01, 1960) represents the month Jan 1960 and (id2, 2, NULL, NULL, 1960) represents the year 1960. The second attribute represents the new attribute *level*: 0 for days, 1 for months, 2 for years. In the fact table, a record containing the foreign key id2 represents the aggregated sales for a Product in the year 1960. The latter method, while reducing the number of tables, is often a source of operational errors since the level field needs be carefully interpreted.

VI. METADATA AND WAREHOUSE MANAGEMENT

Since a data warehouse reflects the business model of an enterprise, an essential element of a warehousing architecture is metadata management. Many different kinds of metadata have to be managed. *Administrative* metadata includes all of the information necessary for setting up and using a warehouse: descriptions of the source databases, back-end and front-end tools; definitions of the warehouse schema, derived data, dimensions and hierarchies, predefined queries and reports; data mart locations and contents; physical organization such as data partitions; data extraction, cleaning, and transformation rules; data refresh and purging policies; and user profiles, user authorization and access control policies. *Business* metadata includes business terms and definitions, ownership of the data, and charging policies. *Operational* metadata includes information that is collected during the operation of the warehouse: the lineage of migrated and transformed data; the currency of data in the warehouse (active, archived or purged); and monitoring information such as usage statistics, error reports, and audit trails.

Often, a metadata repository is used to store and manage all the metadata associated with the warehouse. The repository enables the sharing of metadata among tools and processes for designing, setting up, using, operating, and administering a warehouse. Commercial examples include Platinum Repository and Prism Directory Manager.

Creating and managing a warehousing system is hard. Many different classes of tools are available to facilitate different aspects of the process described in Section 2. Development tools are used to design and edit schemas, views, scripts, rules, queries, and reports. Planning and analysis tools are used for what-if scenarios such as understanding the impact of schema changes or refresh rates, and for doing capacity planning. Warehouse management tools (e.g., HP Intelligent Warehouse Advisor, IBM Data Hub, Prism Warehouse Manager) are used for monitoring a warehouse, reporting statistics and making suggestions to the administrator: usage of partitions and summary tables, query execution times, types and frequencies of drill downs or rollups, which users or groups request which data, peak and average workloads over time, exception reporting, detecting runaway queries, and other quality of service metrics. System and network management tools (e.g., HP OpenView, IBM NetView, and Tivoli) are used to measure traffic between clients and servers, between warehouse servers and operational databases, and so on. Finally, only recently have workflow management tools been considered for managing the extract scrub- transform-load-refresh process. The steps of the process can invoke appropriate scripts stored in the repository, and can be launched periodically, on demand, or when specified events occur. The workflow engine ensures successful completion of the process, persistently records the success or failure of each step, and provides failure recovery with partial roll back, retry, or roll forward.

VII. RESEARCH ISSUES

Data cleaning is a problem that is reminiscent of heterogeneous data integration, a problem that has been studied for many years. But here the emphasis is on *data* inconsistencies instead of schema inconsistencies. Data cleaning, as we indicated, is also closely related to data mining, with the objective of suggesting possible inconsistencies.

The problem of physical design of data warehouses should rekindle interest in the well-known problems of index selection, data partitioning and the selection of materialized views. However, while revisiting these problems, it is important to recognize the special role played by aggregation.

Decision support systems already provide the field of query optimization with increasing challenges in the traditional questions of selectivity estimation and cost-based algorithms that can exploit transformations without exploding the search space (there are plenty of transformations, but few reliable cost estimation

techniques and few smart cost-based algorithms/search strategies to exploit them). Partitioning the functionality of the query engine between the middleware (e.g., ROLAP layer) and the back end server is also an interesting problem.

The management of data warehouses also presents new challenges. Detecting runaway queries, and managing and scheduling resources are problems that are important but have not been well solved. Some work has been done on the logical correctness of incrementally updating materialized views, but the performance, scalability, and recoverability properties of these techniques have not been investigated. In particular, failure and check pointing issues in load and refresh in the presence of many indices and materialized views needs further research. The adaptation and use of workflow technology might help, but this needs further investigation.

REFERENCES

[1] Inmon, W.H., *Building the Data Warehouse*. John Wiley, 1992.

[2] http://www.olapcouncil.org

[3] Codd, E.F., S.B. Codd, C.T. Salley, "Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate."

[4] http://pwp.starnetinc.com/larryg/articles.html

[5] Kimball, R. *The Data Warehouse Toolkit*. John Wiley, 1996.

[6] Barclay, T., R. Barnes, J. Gray, P. Sundaresan, "Loading Databases using Dataflow Parallelism." *SIGMOD Record*, Vol. 23, No. 4, Dec.1994.

[7] Blakeley, J.A., N. Coburn, P. Larson. "Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates." *ACM TODS*, Vol. 4, No. 3, 1989.

[8] Gupta, A., I.S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications." *Data Eng. Bulletin*, Vol. 18, No. 2, June 1995.

[9] Zhuge, Y., H. Garcia-Molina, J. Hammer, J. Widom, "View Maintenance in a Warehousing Environment, *Proc. Of SIGMOD Conf.*, 1995.

[10] Roussopoulos, N., et al., "The Maryland ADMS Project: Views R Us*." Data Eng. Bulletin*, Vol. 18, No.2, June 1995.

[11] O'Neil P., Quass D. "Improved Query Performance with Variant Indices", to appear in *Proc. of SIGMOD Conf.*, 1997.

[12] O'Neil P., Graefe G. "Multi-Table Joins through Bitmapped Join Indices" *SIGMOD Record*, Sep 1995.

[13] Harinarayan V., Rajaraman A., Ullman J.D. "Implementing Data Cubes Efficiently" *Proc. of SIGMOD Conf.*, 1996.

[14] Chaudhuri S., Krishnamurthy R., Potamianos S., Shim K. "Optimizing Queries with Materialized Views" *Intl. Conference on Data Engineering*, 1995.

[15] Chaudhuri S., Dayal U. *"An Overview of Data Warehousing and OLAP Technology*" ACM Sigmod Record, March 1997.