ADVANCES AND RESEARCH DIRECTIONS IN DATA WAREHOUSING TECHNOLOGY

Mukesh Mohania
School of Computer and Information Science
University of South Australia
Mawson Lakes, Adelaide.
South Australia 5095
mohania@cis.unisa.edu.au

Sunil Samtani
Department of Computer Science Telecommunications,
University of Missouri-Kansas City,
Kansas City, MO 64110, U.S.A.
ssamtani@cstp.umkc.edu

John F. Roddick
School of Computer and Information Science
University of South Australia
Mawson Lakes, Adelaide.
South Australia 5095
roddick@cis.unisa.edu.au

Yahiko Kambayashi
Department of Social Informatics,
Graduate School of Informatics,
Kyoto University,
Kyoto 606-8501, Japan.
yahiko@kuis.kyoto-u.ac.jp

ABSTRACT

Information is one of the most valuable assets of an organisation and when used properly can assist in intelligent decision making that can significantly improve the functioning of an organisation. Data Warehousing is a recent technology that allows information to be easily and efficiently accessed for decision-making activities by collecting data from many operational, legacy and possibly heterogeneous data sources. On-Line Analytical Processing (OLAP) tools are well-suited for complex data analysis, such as multi-dimensional data analysis, and to assist in decision support activities while data mining tools take the process one step further and actively search the data for patterns and hidden knowledge in the data held in the warehouse. Many organisations are building, or are planning to develop, a data warehouse for their operational and decision support needs. In this paper, we present an overview of data warehousing, multi-dimensional databases, OLAP and data mining technology and discuss the directions of current research in the area. We also discuss recent developments in data warehouse modelling, view selection and maintenance, indexing schemes, parallel query processing and data mining issues. A number of technical issues for exploratory research are presented and possible solutions are also discussed.

INTRODUCTION

Relational database systems are designed to record, retrieve and manage large volumes of real-time transaction data and to keep an organisation running by supporting day-to-day business transaction processing. These systems are generally tuned for a large community of users and the user typically knows what is needed and generally accesses a small number of rows in a single transaction. Indeed, relational database systems are suited for robust and efficient On-Line Transaction Processing (OLTP) on operational data. One of the main objectives of relational systems is to maximize transaction throughput and minimize concurrency conflicts. However, these systems have generally limited decision support functions and do not possess or enable the extraction of all the necessary information required for faster and more intelligent decision-making essential for the growth of an organisation. For example, it is hard for a relational database system to answer queries such as *what were the supply patterns of product x in South Australia in 1997 and how were they different from the previous year?* It has therefore become important to support analytical processing capabilities in organisations for (1) the efficient management of organisations, (2) effective marketing strategies and (3) efficient and intelligent decision-making.

On-Line Analytical Processing (OLAP) tools are well-suited for complex data analysis, such as multi-dimensional data analysis, and to assist in decision support activities which access data from a separate *data warehouse*, which may consist of data collected from many, possibly heterogeneous, operational and legacy data sources. The data warehouse system is typically tuned for a smaller community of users including business and technical managers, directors and/or decision-makers.

A data warehouse is a set of *subject-oriented*, *integrated*, *time-varying* and *non-volatile* databases used to support the decision-making activities (Inmon 1992). The term subject-oriented indicates that information is arranged according to the business areas of an organisation, such as manufacturing, sales, marketing, finance, etc. The term integrated refers to the fact that a data warehouse integrates data derived from various functional

systems in the organisation and provides a unified and consistent view of the overall organisational situation. Time-variant means that warehouse data is able to represent the flow of data through time, and queries may be based on time ranges. The data is non-volatile in that it is always appended to (and rarely deleted from) the warehouse thus maintaining the company's entire history. Thus, a data warehouse system is a repository system that contains highly aggregated and summarised data used for complex data analysis and decision support activities, such as data mining, comparisons of historical data and trend analysis. For efficient and correct analysis, the quality of data is important, that is,

- the warehouse data should be accurate, complete, consistent, integrated, well-defined and time-stamped for informational purposes,
- the warehouse data should conform to appropriate business rules and satisfy integrity constraints, including primary/foreign keys, referential integrity and data dependencies,
- users, including the auditors, should be satisfied with the quality of data and information derived from warehouse.

Typical warehouse queries are complex and ad-hoc in nature and generally these queries access huge volumes of warehouse data and perform many joins and aggregations. Query response time and throughput are therefore more important than transaction throughput. In summary, the data warehousing environment provides a computerised interface that enables business decision-makers to creatively approach, analyse and understand business problems. The aim of the data warehouse system is to turn data into strategic decision making information and to bring solutions to users. This process is done by tuning the data at many steps.

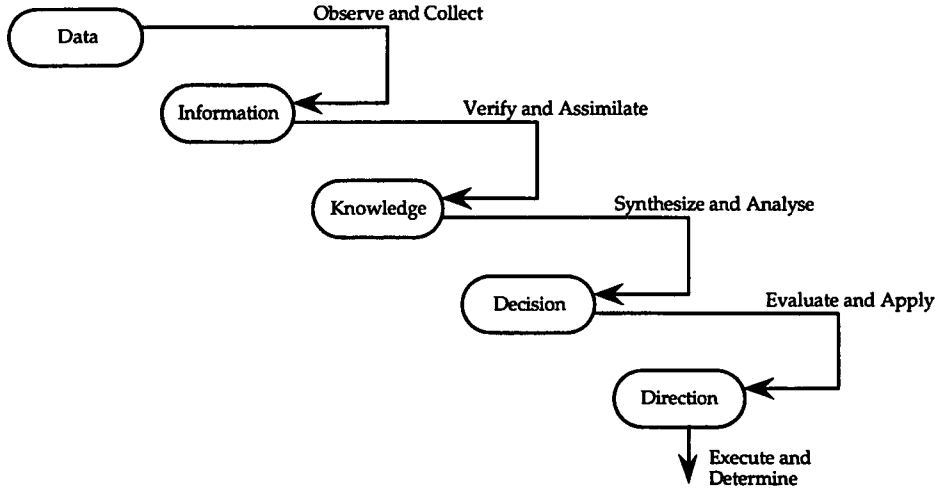

Figure 1. Data Analysis Chart

Data Warehousing System

The data warehouse creation and management component includes software tools for selecting data from appropriate information sources (which could be operational, legacy, external, etc. and which may be distributed, autonomous and heterogeneous), and cleaning, transforming, integrating and propagating data into the data warehouse. It also refreshes the warehouse data and metadata when source data are updated. This component is also responsible for managing the warehouse data, creating indices on data tables and data partitioning. The warehouse data contains the detail data, summary data, consolidated data and/or multi-dimensional data. The metadata is generally held in a separate repository. The metadata contains the informational data about the creation, management and usage of the data warehouse. It serves as a bridge between the users of the warehouse and the data contained in it. The warehouse data is also accessed by the OLAP server to present the data in a multi-dimensional way to the front-end tools (such as analytical tools, report writers, spreadsheets and data mining tools) for analysis and informational purposes. Essentially, the OLAP server interprets client queries (the client interacts with front-end tools and passes these queries to the OLAP server) and converts them into complex SQL (indeed extended SQL) queries required to access the warehouse data. It might also access the data from the primary sources if the client's queries need operational data. Finally, the OLAP server passes the multi-dimensional views of data to the front-end tools and these tools format the data according to the client's requirements. The conceptual architecture of a data warehousing system is shown in Figure 2.

There are two approaches to creating the warehouse data — bottom-up and top-down. In a bottom-up approach, the data is obtained from the primary sources based on the data warehouse applications and a profile of the likely queries, which is typically known in advance. The data is then selected, transformed and integrated by

data acquisition tools. In a top-down approach, the data is obtained from the primary sources whenever a query is posed. In this case, the warehouse system determines the primary data sources in order to answer the query. These two approaches are similar to *eager* and *lazy* approaches discussed by Widom (1995). The bottom-up approach is used in data warehousing because user queries can be answered immediately and data analysis can be done efficiently since data will always be available in the warehouse. Hence, this approach is feasible and improves the performance of the system. Another approach is a hybrid approach, which combines aspects of the bottom-up and top-down approaches. In this approach, some data is stored in a warehouse, and other data can be obtained from the primary sources on demand (Hull and Zhou 1996).

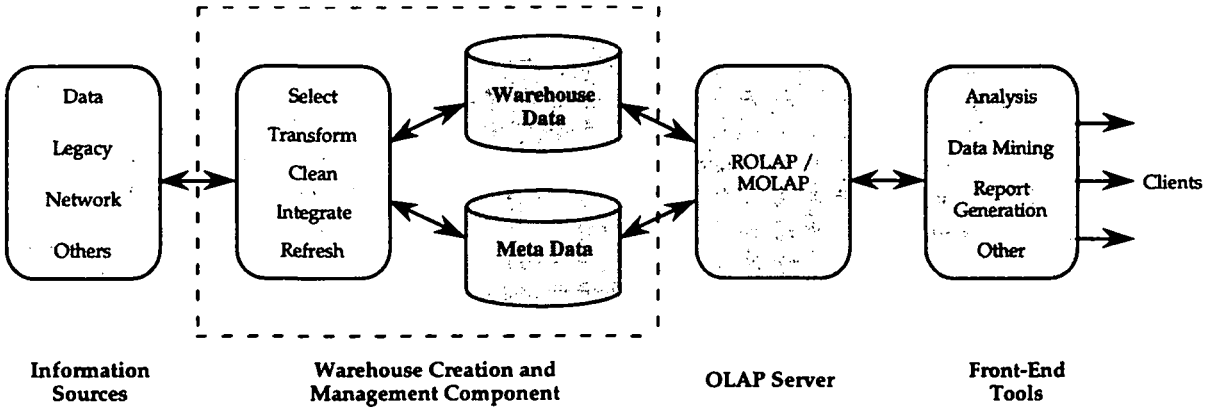

Figure 2. A conceptual data warehousing architecture

Motivation

Widom (1995) outlined a number of technical issues in data warehousing which warranted the attention of the research community. A lot of work has been done in many areas since then and new issues have come to the fore. Different aspects of data warehousing present different related and unrelated challenges with the common goal of presenting integrated information to the user quickly. Different methodologies have been used to achieve this goal. Thus, data warehousing has evolved into a number of different research streams. These streams deal with different phases of data warehousing - design, implementation, operation and maintenance. In this paper, we explore the different aspects of this emerging technology and highlight new avenues still unexplored.

Chaudhuri and Dayal (1997) present the state of art in data warehousing and OLAP with an emphasis on new requirements. The work presented serves as a good overview and is an excellent survey in this area. However, there is a need to initiate a discussion on the impending research problems in the field. In this paper, we suggest new directions to some of these problems by providing insight for developing efficient solutions and by discussing some of the well known solutions.

Structure of this Paper

The warehouse data is typically modelled *multi-dimensionally*. New data models and new data structures that support the multi-dimensional data model effectively needs to be developed. The multi-dimensional data model (Agrawal, Gupta and Sarawagi 1997; Lehner, Ruf and Teschke 1996) has been proved to be the most suitable for OLAP (On-Line Analytical Processing) applications and OLAP tools provide an environment for decision making and business modelling activities by supporting ad-hoc queries. There are two ways to implement multi-dimensional data model:

- by using the underlying relational architecture to project a pseudo-multi-dimensional model and
- by using true multi-dimensional data structures such as arrays.

We discuss the multi-dimensional model and the implementation schemes in the next section. Although the idea of a true *multi-dimensional* implementation is lucrative theoretically, it has not been widely investigated by either the research or commercial communities. However, some work has been done in defining a new algebra for multi-dimensional data model and new algebraic operations and extensions to existing SQL have been proposed in the literature. Nevertheless, the use of this multi-dimensional data model has not been exploited for all aspects of data warehouse design and maintenance. The basic feature of this model is that it allows the user to visualise data from different perspectives. There is a need to extend its capabilities by defining a set of constraints on this model so that they can be exploited in other areas of warehouse implementation. This paper contributes in a useful way by setting the platform for extending the multi-dimensional data model by

introducing the concept of constraints on the data cube. We also highlight the need for constraints in the multi-dimensional environment and its proposed uses.

The multi-dimensional data model is stored in the data warehouse as a set of materialized views over source data. There are limitations to the concept of *materializing* views at the data warehouse. Precomputation of queries in materialized views can give answers quickly but the number of views that should be materialized at the warehouse needs to be controlled, otherwise this can result in *data explosion*. We investigate the issue of the selection of views to be materialized at the data warehouse and describe well-known heuristics in detail.

One challenge in data warehousing is how to maintain the materialized views. When the data at any source changes, the materialized views at the data warehouse need to be updated accordingly. The process of keeping the views up-to-date in response to the changes in the source data is referred to as *View Maintenance*. Refreshing the warehouse data is often done, for reasons of efficiency, using incremental techniques rather than recomputing the view from scratch. There has been significant research on view maintenance in traditional databases (Bailey, *et al.* 1998; Blakeley, Larson and Tompa 1986; Dong and Mohania 1996) (Griffin and Libkin 1995; Gupta and Mumick 1995; Gupta, Mumick and Subrahmanian 1993; Ramakrishnan, *et al.* 1994; Segev and Fang 1990; Segev and Park 1989) and in data warehouses (Agrawal, *et al.* 1997; Hull and Zhou 1996; Huyn 1997; Quass, *et al.* 1996; Yang, Karlapalem and Li 1996) (Zhuge, *et al.* 1995). In data warehousing, the view maintenance has branched into a number of sub-problems such as self-maintenance, consistency maintenance, update filtering and on-line view maintenance. We investigate these issues and propose new ways of looking at some of these problems.

The technique of view materialization is hampered by the fact that one needs to anticipate the queries to materialize at the warehouse. The queries issued at the data warehouse are mostly *ad-hoc* and cannot be effectively anticipated at all times. Users might query on dimensions that are not materialized in views. Thus, the precomputation strategy is limited in its ways. Effective translation of data cubes in the relational model results in star/snowflake schema. These require effective indexing methods since they involve joins on multiple tables for answering queries. The query is not precomputed and needs to be computed *on the fly*. The data is integrated from various sources and stored in the data warehouse *in advance* to answer decision support queries from the user. This needs faster access structures to keep the processing on-line. The traditional indexing methods in relational databases do not work well in the data warehousing environment and new access structures have been proposed for data warehousing environments. We investigate different types of indexing schemes in Section 4. A relatively new approach to compute expensive operations such as join and aggregation is through the use of parallel processing techniques. We discuss the issues of parallelism in data warehousing in Section 5. The accumulation of data in a data warehouse provides opportunities for data mining and issues relating to mining from data warehousing are discussed. Finally, we comment on other areas of active research and conclude the paper with a discussion on possible future work.

DATA MODELS FOR DATA WAREHOUSING

The data models for designing traditional OLTP systems are not well suited for modelling complex queries in data warehousing environment. The transactions in OLTP systems are made up of simple, pre-defined queries. In the data warehousing environments, the queries tend to use joins on more tables, have a larger computation time and are ad-hoc in nature. This kind of processing environment warrants a new perspective to data modelling. The *multi-dimensional* data model, sometimes referred to as the *data cube* has turned out to be a satisfactory model that provides a way to aggregate facts along multiple attributes, called *dimensions*. Data is stored as *facts* and *dimensions* instead of rows and columns as in relational data model. Facts are numeric or factual data that represent a specific business activity and a dimension represent a single perspective on the data. Each dimension is described by a set of attributes.

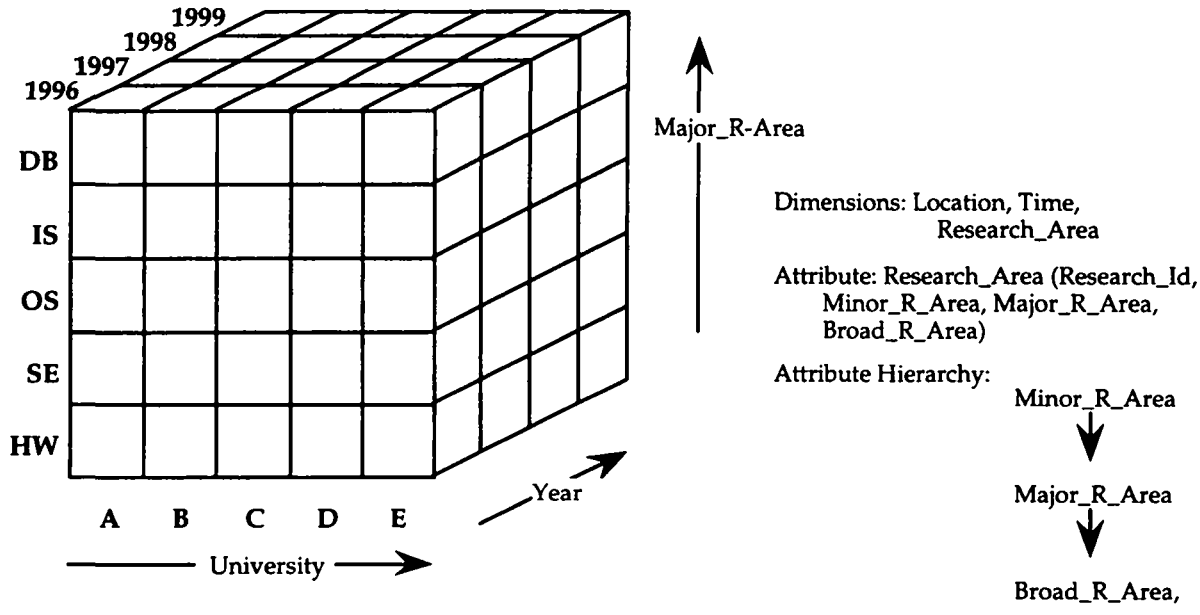

Figure 3. Data Cube

A multi-dimensional data model (MDDM) supports complex decision queries on huge amounts of enterprise and temporal data. It provides an integrated environment for summarising (using aggregate functions or by applying some formulae) information across multiple dimensions. MDDM has now become the preferred choice of many vendors as the platform for building new on-line analytical processing (OLAP) tools. The user has the leverage to *slice* and *dice* the dimensions, allowing users to employ different dimensions during an interactive query session. The data cube allows the user to visualise aggregated facts multi-dimensionally. The level of detail retrieved depends on the number of dimensions used in the data cube. When the data cube has more than three dimensions, then it is called a *hypercube*. The dimensions form the axes of the hypercube and the solution space represents the facts as aggregates on measure attributes. The cube for *publications* data is shown in the figure below.

Implementation Schemes

The conceptual multi-dimensional data model can be physically realised in two ways, firstly by using trusted relational databases and secondly by making use of specialised multi-dimensional databases. Each approach is briefly described below.

Relational Scheme

This scheme stores the data in specialised relational tables, called fact and dimension tables. It provides a multi-dimensional view of the data by using relational technology as an underlying data model. Facts are stored in the fact table and dimensions are stored in the dimension table. For example, the *Research_Area* dimension consists of four attributes, namely, *Research_Id*, *Minor_R_Area*, *Major_R_Area* and *Broad_R_Area*. Examples of fact tables include Sales data, Publication data, etc, and the examples of dimension tables include products, stores, documents, research areas, etc. Facts in the fact table are linked through their dimensions. The attributes that are stored in the dimension table may exhibit attribute hierarchy.

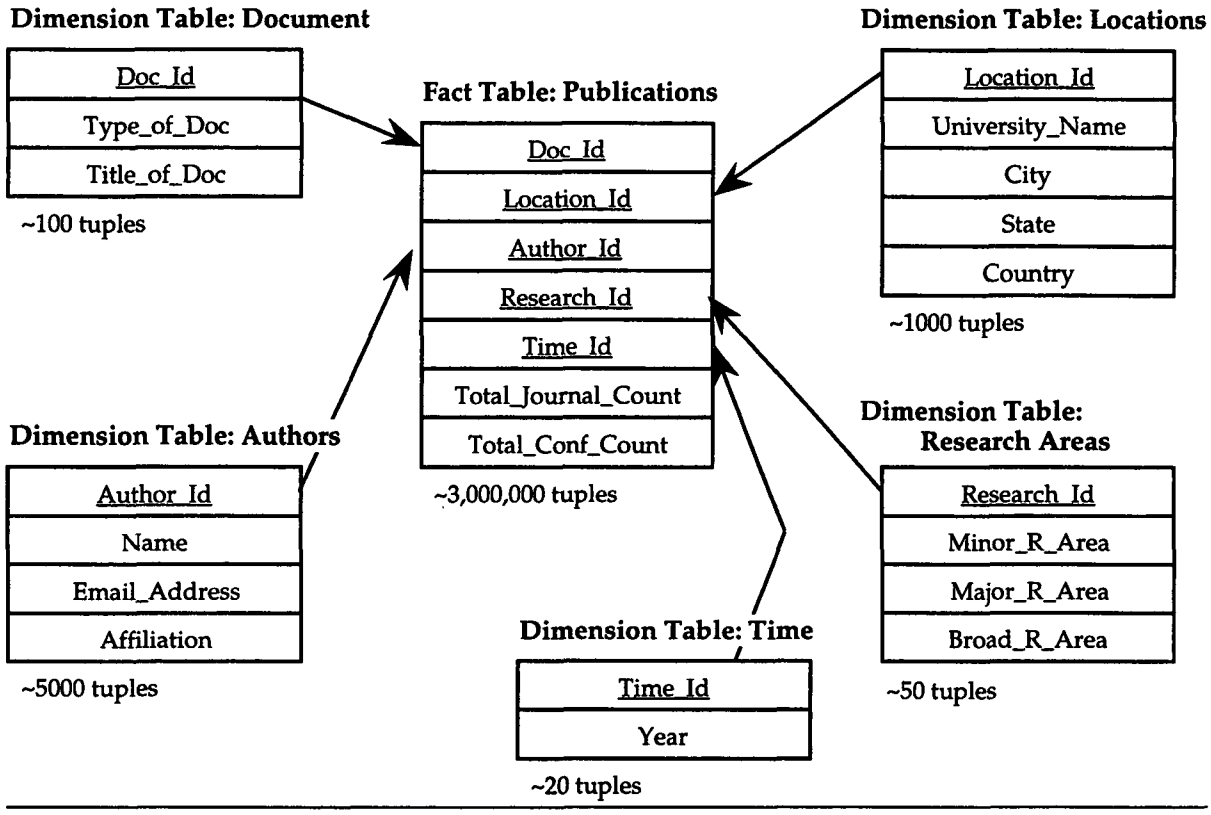

Figure 4. Star Schema Example

Star schema/snowflake schema are used to support multi-dimensional data representation. They offer flexibility, but often at the cost of performance because of more joins for each query required. A star/snowflake schema models a consistent set of facts (aggregated) in a fact table and the descriptive attributes about the facts are stored in multiple dimension tables. This schema makes heavy use of denormalization to optimise complex aggregate query processing. In a star schema, a single fact table is related to each dimension table in a many-to-one (M:1) relationship. Each dimension tuple is pointed to many fact tuples. Dimension tables are joined to fact table through foreign key reference; there is a referential integrity constraints between fact table and dimension table. The primary key of the fact table is a combination of the primary keys of dimension tables. Note that multiple fact tables can be related to the same dimension table and the size of dimension table is very small as compared to the fact table. The figure below shows an example of star schema. Here, publication data is stored in fact table and location, document, authors, research area and time are dimension tables.

As can be seen above, the dimension tables are denormalized and therefore, the star schema does not capture hierarchies (ie. dependencies among attributes) directly. This is captured in *snowflake schema*. Here, the dimension tables are normalised for simplifying the data selecting operations related to the dimensions, and thereby, capture attribute hierarchies. In this schema, the multiple fact tables are created for different aggregate levels by pre-computing aggregate values. This schema projects better semantic representation of business dimensions.

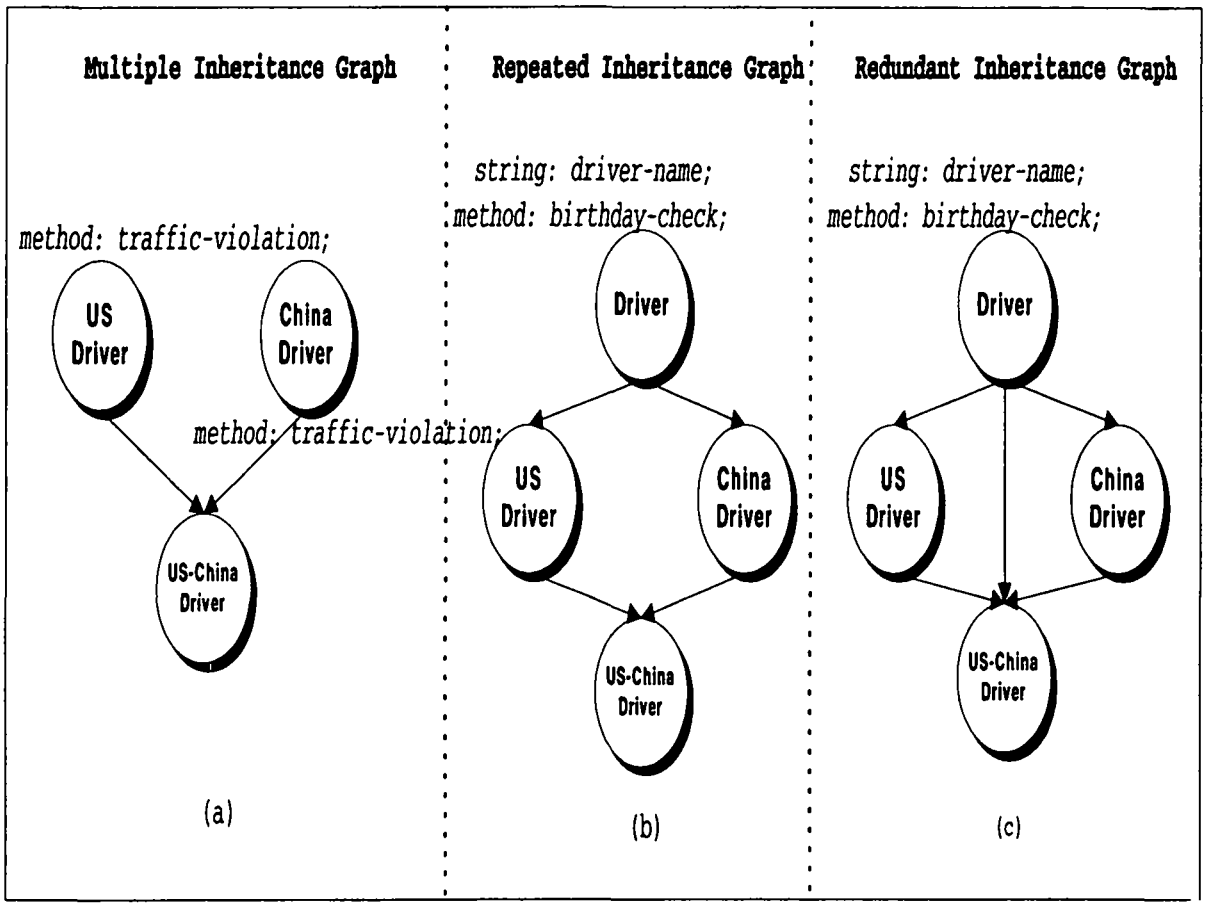

Figure 5. Snowflake Schema Example

Multi-dimensional data model which translates the data cube into a star/snowflake schema is defined as follows (Baralis, Paraboschi and Teniente 1997): A **Multi-dimensional Database** is a collection of *n* hypercubes $\{H_1, H_2,..., H_n\}$, where each hypercube H_i is a set of relations $\{D_1, D_2, ..., D_m, F\}$ such that

- Each D_j is a **dimension table**, ie. a relation characterised by its unique identifier d_j (dimension) that uniquely identifies each tuple in D_j and provides the granularity for representing facts along the axis of the hypercube.
- F is a **fact table**, ie., a relation connecting all dimension tables $D_1, D_2, ..., D_m$ on the dimension attribute d_j in each table; the identifier of F is given by the foreign keys $d_1, ..., d_m$ of all the dimension tables it connects; the schema of F contains a set of additional attributes M called measure attributes on which the aggregate functions are computed; the aggregate functions on M form the solution space of the hypercube.

Each dimension table may contain either a simple dimension or a composite dimension.. A composite dimension is a collection of attributes that exhibit certain functional dependencies that gives rise to dimension attribute hierarchies.

Let d be a composite dimension in dimension table D such that schema(d) = $\{A_1, ..., A_n\}$. The **attribute hierarchy** on D is defined as set of functional dependencies $FD_D = \{fd_1, fd_2, ..., fd_n\}$ where each fd_i represents a functional dependency from A_i to A_j; the dependency is represented by fd_i: $A_i \rightarrow A_j$.

The hypercubes are materialized as views at the data warehouse. The materialized views that store the preprocessed queries at the data warehouse along different dimensions are selected to minimize the query response time and the view update costs. Another criterion for selecting a view in a data warehouse is its frequency of access. A query might not be accessed frequently but it should be materialized in a view at the data warehouse (DW) if other queries that are accessed frequently can be answered quickly using this view. This is an area of active research and several heuristics are proposed (Baralis, Ceri and Paraboschi 1996; Baralis, Paraboschi and Teniente 1997; Labio, Quass and Adelberg 1997; Ross, Srivastava and Sudarshan 1996). The problem of view maintenance and the problem of view selection is discussed further later.

Multi-dimensional Scheme

This scheme stores data in a matrix using array-based storage structure. Each cell in the array is formed by the intersection of all the dimensions; therefore, not all cells have a value. For example, not all authors publish in all journals. The multi-dimensional data set requires smaller data storage since the data is clustered compactly in the multi-dimensional array. The values of the dimensions need not be explicitly stored. The *n*-dimensional table schema is used to support multi-dimensional data representation, which is now described.
An *n*-dimensional table schema is the fundamental structure of a multi-dimensional database that draws on terminology of the statistical databases. The attribute set associated with this schema is of two kinds: *parameters* and *measures*. An *n*-dimensional table has a set of attributes R and a set of dimensions D associated with it. Each dimension is characterised by a distinct subset of attributes from R, called the parameters of that dimension. The attributes in R, which are not parameters of any dimension, are called the measure attributes. This approach is a unique way of *flattening* the data cube since the table structure is inherently multi-dimensional. The actual contents of the table are essentially orthogonal to the associated structure. Each *cell* of the data cube can be represented in an *n*-dimensional table as table entries. These table entries have to be extended by dimensions to interpret their meaning. The current literature on *n*-dimensional table however does not give an implementation of the MDDB that is different from the implementation suggested by the already existing schemas. This implementation breaks up the *n*-dimensional table into dimension tables and fact tables, which snowballs into snowflake schema and traditional ROLAP. The challenge with the research community is to find mechanisms that translate this multi-dimensional table into a true multi-dimensional implementation. This would require new data structures for the implementation of multiple dimensions in one table. The relation in the relational data model is an example of 0-dimensional table.

Constraints on the Cube Model

In a relational schema, it is possible to define a number of integrity constraints in the conceptual design. These constraints can be broadly classified as key constraints, referential integrity constraints, not null constraint, relation-based check constraints, attribute-based check constraints and general assertions (business rules). These constraints can be easily translated into triggers that keep the relational database consistent at all times. This concept of defining constraints based on dependencies can be mapped to a multi-dimensional scenario.
The current literature on modelling multi-dimensional databases has not discussed the constraints on the data cube. In a relational model, the integrity and business constraints that are defined in the conceptual schema provide for efficient design, implementation and maintenance of the database. Taking a cue from the relational model, it is necessary to identify and enumerate the constraints that exist in the multi-dimensional model. An exploratory research area would be to categorise the cube constraints into classes and compare them with the relational constraints. The constraints can be broadly classified into two categories: *intra-cube* constraints and *inter-cube* constraints. The intra-cube constraints define constraints within a cube by exploiting the relationships that exist between the various attributes of a cube. The relationships between the various dimensions in a cube, the relationships between the dimensions and measure attributes in a cube, dimension attribute hierarchy and other cell characteristics are some of the key cube features that need to formalised as a set of intra-cube constraints. The inter-cube constraints define relationships between two or more cubes. There are various considerations in defining inter-cube constraints. Such constraints can be defined by considering the relationships between dimensions in different cubes, the relationships between measures in different cubes, the relationships between measures in one cube and dimensions in the other cube and the overall relationship between two cubes, ie. two cubes might merge into one, one cube might be a subset of the other cube, etc.

Using Constraints

The cube constraints facilitate the conceptual schema design of a data warehouse and allow the provision of triggers in a multi-dimensional scenario. In a relational database system, triggers play an important role by enforcing the constraints and business rules in an effective manner. The UPDATE, INSERT and DELETE triggers in the relational model have provided the robustness by allowing least manual intervention. The current data warehouse maintenance algorithms have neglected the role of triggers in designing effective view maintenance paradigms. The research community needs to consider the new types of triggers that could be used for effective data warehousing. The constraint set will be of vital importance to this faculty.
The constraint set can also be used to solve existing problems such as view maintenance. In (Mohania, Konomi and Kambayashi 1997; Ross, Srivastava and Sudarshan 1996), the authors have proposed view maintenance techniques based on the view expression tree that is used for evaluating the view as a query. The auxiliary

relations are materialized for each node in the tree in addition to the materialized view. The updates to nodes in the expression tree are calculated and propagated in a bottom-up fashion. The update to each node in the tree is derived from the updates to its children nodes and the auxiliary relations materialized for the children and the node itself. If the constraints are defined on the view, then they can be broken down into smaller components (ie. sub-constraints) and these sub-constraints can be pushed down into the expression tree to provide new efficient mechanisms for view maintenance and aggregate query optimisation. That is, the constraints can be defined at each node in the tree. In this case, the updates to each node can be checked using these constraints before propagating them to the parent node. If some of the updates violate constraints, then there is no need to propagate these updates to the parent node. This method of decomposing the constraints and pushing them down along the edges of the expression tree can effectively reduce the communication and computation costs. The constraint set acts as a sieve that effectively filters the updates that do not affect the view.
Furthermore, the constraint set will be an effective tool that can be used in other aspects of data warehousing. For example, the constraint set would enable the definition of a complete algebra for the multi-dimensional data model that is independent of the underlying schema definitions. The constraint set would also allow evolution of new attributes in the multi-dimensional scenario. Some of these constraints can be used to map the multi-dimensional world into the relational world. The data model can be viewed as a ball and the constraints can help visualise that ball from various perspectives. The constraint set would enable new implementation mechanisms that are better able to conserve the multi-dimensionality of data. The first task is to define a constraint set on the multi-dimensional data model that is complete in any implementation scheme. The constraints need to be formally defined and the precedence order for the constraints needs to be decided.

Operations in Multi-dimensional Data Model

Data warehousing query operations include standard SQL operations, such as selection, projection and join. In addition, it supports various extensions to aggregate functions, for example, percentile functions (eg. top 20 percentile of all products), rank functions (eg. top 10 products), mean, mode and median. One of the important extension to the existing query language is to support multiple group by by defining *rollup*, *drill-down* and *cube* operators. Rollup corresponds to doing further group-bys on the same data object. For example, let the publications be grouped by minor-research area. It is possible to further group-by the same publications data by major-research area and broad-research area by using rollup operation. Note that rollup operator is order sensitive, that is, when it is defined in the extended SQL, the order of columns (attributes) matters. The function of drill-down operation is the opposite of rollup. The hypercube which involves joining of multiple tables to represent facts needs a new set of algebraic operations. A new algebra needs to be proposed for the multi-dimensional environment. The idea of *faster* query processing will require extensions to existing SQL in the existing environment. New operators such as *cube*, *push*, *pull*, *restrict*, *star*, *join* and *merge* have been proposed in literature but all these operators are specific to the schema for which they are designed (Agrawal, Gupta and Sarawagi 1997; Bauer and Lehner 1997; Lehner, Ruf and Teschke 1996; Li and Wang 1996).

VIEW SELECTION AND MAINTENANCE

View Selection

We discussed earlier which views should be materialized at the data warehouse. The problem of view selection was first studied by Roussopoulos (Roussopoulos 1982). Recently, several heuristics have been proposed for the same problem (Baralis, Ceri and Paraboschi 1996; Baralis, Paraboschi and Teniente 1997; Labio, Quass and Adelberg 1997; Ross, Srivastava and Sudarshan 1996)}. In (Roussopoulos 1982), an algorithm based on A^* and the approximate knapsack problem has been proposed. This algorithm does not consider the cost of subviews to materialise and index selection. (Ross, Srivastava and Sudarshan 1996) describes an exhaustive enumerative search algorithm where they consider the cost of subviews to materialise, but without considering indexes. (Labio, Quass and Adelberg 1997) has considered both the cost of subviews and indexing in their optimal view selection algorithm. We now discuss a heuristic (Baralis, Paraboschi and Teniente 1997) in detail which uses the data cube technology and the lattice model. The lattice model feeds on the *attribute hierarchy* defined earlier. The nodes in the *lattice* diagram represent views (aggregated on certain dimensions) to be materialized at the data warehouse. If the dimensions of two views *a* and *b* exhibit attribute hierarchy such that *dim(a)* $\rightarrow$ *dim(b)* then there is an edge from node *a* to node *b*. Node *a* is called the *ancestor* node and node *b* is called the *dependent* or *descendant* node. The lattice diagram allows us to establish relationships between views that need to be materialized. Some queries can be answered by using the already materialized views at the data warehouse. For answering such queries, it is not necessary to use the raw data. The lattice diagram depicts dependencies between the views and a good view selection heuristic exploits this dependency. The view selection algorithm tries to minimize the average time required to evaluate a view and also keeps a constraint on

the space requirements. The space requirements that can be expressed as the number of views to be materialized translates this into an optimisation problem that is NP-complete. An approximate and acceptable solution for this problem is the *greedy* heuristic. The greedy algorithm selects a view from a set of views depending upon the benefit yielded on selecting that view. A view *a* that is materialized from view *b* incurs a materializing cost that is equal to the number of rows in view *b*. If there is a view *c* (materialized on *b*; number of rows in *c* • number of rows in *b*) such that view *a* can be derived from *c*, the cost of materializing *a* reduces to the number of rows in *c*. Thus the benefit of materializing view *c* includes the benefit incurred by view *a* in the form of reduction in its materializing cost (number of rows in *b* - number of rows in *c*). The greedy heuristic selects a view that maximizes the benefit that will be yielded on materializing that view. The benefit of materializing each *dependent* view (a node in the lattice diagram) will change with the selection of an *ancestor* view in the data warehouse. After each selection is made, the benefit at each *dependent* node in the lattice is recalculated and a view with maximum benefit is selected. It has been shown that the greedy algorithm is at least 3/4 of optimal. The greedy algorithm can be extended to restrict on actual space requirements rather than the number of views to be materialized. The frequency with which the views are accessed can be incorporated in this algorithm. Other criteria for selecting views may also be based on the way the views are implemented at the warehouse. Different schema design methods will give different set of tables at the data warehouse.

View Maintenance

A data warehouse (DW) stores integrated information from multiple data sources in *materialized views* (*MV*) over the source data. The data sources (DS) may be heterogeneous, distributed and autonomous. When the data in any source (base data) changes, the *MV*s at the *DW* need to be updated accordingly. The process of updating a materialized view in response to the changes in the underlying source data is called View Maintenance. The view maintenance problem has evoked great interest in the past few years. This view maintenance in such a distributed environment gives rise to inconsistencies since there is a finite unpredictable amount of time required for (a) propagating changes from the DS to the *DW* and (b) computing view updates in response to these changes. Data consistency can be maintained at the data warehouse by performing the following steps:

- propagate changes from the data sources (ST_1 - current state of the data sources at the time of propagation of these changes) to the data warehouse to ensure that each view reflects a consistent state of the base data.
- compute view updates in response to these changes using the state ST_1 of the data sources.
- install the view updates at the data warehouse in the same order as the changes have occurred at the data sources.

The inconsistencies at the data warehouse occur since the changes that take place at the data sources are random and dynamic. Before the data warehouse is able to compute the view update for the *old* changes, the *new* changes change the state of the data sources from ST_1 to ST_2. This violates the consistency criterion that were listed. Making the *MV*s at the data warehouse self-maintainable decimates the problem of inconsistencies by eliminating the finite unpredictable time required to query the data source for computing the view updates. In the next subsection, the self-maintenance of materialized views at the data warehouse is described.

Self-Maintenance

Consider a materialized view *MV* at the data warehouse defined over a set of base relations $R = \{R_1, R_2, ..., R_n\}$. *MV* stores a preprocessed query at the data warehouse. The set of base relations *R* may reside in one data source or in multiple, heterogenous data sources. A change ΔR_i made to the relation R_i might affect *MV*. *MV* is defined to be self-maintainable if a change ΔMV in *MV*, in response to the change ΔR_i can be computed using only the *MV* and the update ΔR_i. But the data warehouse might need some additional information from other relations in the set *R* residing in one or more data sources to compute the view update ΔMV. Since the underlying data sources are decoupled from the data warehouse, this requires a finite computation time. Also the random changes at the data sources can give rise to inconsistencies at the data warehouse. Some data sources may not support full database functionalities and querying such sources to compute the view updates might be a cumbersome, even an impossible task. Because of these problems, the preprocessed query that is materialized at the warehouse needs to be maintained without access to the base relations. One of the approaches is to replicate all base data in its entirety at the data warehouse so that maintenance of the *MV* becomes local to the data warehouse (Gupta and Mumick 1995; Gupta, Mumick and Subrahmanian 1993; Kuchenhoff 1991). Although this approach guarantees self-maintainability at the warehouse, it creates new problems. As more and more data is added to the warehouse, it increases the space complexity and gives rise to information redundancy which might lead to inconsistencies. This approach also overlooks the point that the base tuples might be present in the view itself, so the view instance, the base update and a subset of the base relations might be sufficient to achieve self-maintainability in the case of SPJ (Select-Project-Join) views (Huyn 1997). But how can the subset

of the base relations that is needed to compute the view updates be stored at *DW*? This question was addressed in (Quass, *et al.* 1996), which defines a set of minimal *auxiliary views* (*AV*s) to materialize that are sufficient to make a view self-maintainable. Although materializing auxiliary views at the *DW* was a novel concept, the minimality of auxiliary views defined was still questionable since the *MV* instance was never exploited for self-maintenance. Most of the current approaches maintain the *MV*s separately from each other using a separate *view manager* for each view and such approaches fail to recognize that these views can be maintained together by identifying the set of related materialized views. This issue of multiple-view self-maintenance was addressed for the first time in (Huyn 1997).

In some approaches to multiple-view self-maintenance, a set of auxiliary views (*AV*) are stored at the data warehouse along with the set of materialized views (*MV*) such that together $MV \cup AV$ is self-maintainable. The research challenge lies in finding the most *economical AV*s in terms of space complexity and computational costs. The view self maintenance is still an active research problem. It is not always feasible to provide self-maintainability of the views at the data warehouse. When the cost of providing self-maintainability exceeds the cost of querying data sources for computing view updates, it is profitable to allow querying of data sources instead.

Consistency Maintenance

Current research has also concentrated on ensuring consistency of the data warehouse when the *MV*s are not self-maintainable since it is not always possible to provide for complete self-maintainability at the data warehouse. The *ECA* family of algorithms (Zhuge, *et al.* 1995) introduces the problem and solves it partially. The *Strobe* algorithm (Zhuge, Garcia-Molina and Widom 1996) introduces the concept of queuing the view updates in the *Action-List* at the data warehouse and installing the updates only when the *unanswered query set* (UQS) is empty. The algorithm solves the consistency problem but is subject to the potential threat of infinite waiting. There are other mechanisms that are based on time stamping the view updates (Baralis, Ceri and Paraboschi 1996). These methods do not address the consistency problems in their entirety and also assume the notion of global time.

We propose that the self-maintainability of views at the data warehouse should be a dynamic property. The cost of providing self-maintainability should be continuously monitored and when this cost increases beyond a certain threshold, the maintenance mechanism should be shifted to querying data sources to compute the view updates. This threshold can be computed depending on the cost of querying data sources. An effective algorithm that provides this dynamism and efficient garbage collection is the need of the hour.

Update Filtering

The changes that take place in the source data need to be reflected at the data warehouse. Some changes may create view updates that need to installed at the data warehouse; some changes leave the views at the data warehouse unchanged. If it is possible to detect at the data source that certain changes are guaranteed to leave the views unchanged, we need not propagate these changes to the data warehouse. This would require checking of distributed integrity constraints at a single site. As many changes as possible can be filtered at the sources and only the changes that result in view updates may be propagated to the warehouse. The update filtering will reduce the size of the maintenance transactions at the data warehouse, thus minimizing the time required to make the data warehouse consistent with the data sources. The side effect of update filtering is that we need to make our data sources (and the wrapper/monitor) components more intelligent. They need to know about their participation in the data warehouse and the data warehouse configuration so that the updates can be checked against the constraint set before propagating them. To be able to realise this, the data sources cannot be decoupled from the data warehouse. This would give rise to new problems such as configuration management ie., if there is a change in the schema at any data source or at the data warehouse, all the participating entities need to be informed of this change so that they can modify the constraint set to reflect this change. The view maintenance strategies would now be based on the constraint set and any change to the constraint set would warrant a change in the existing view maintenance transaction.

On-Line View Maintenance

Warehouse view maintenance can be performed either *incrementally* or by queuing a large number of updates at the data sources to be propagated as a *batch* update from the data sources to the data warehouse. In current commercial systems, a batch update is periodically sent to the data warehouse and view updates are computed and installed. This transaction is called the *maintenance transaction.* A user typically issues read-only queries at the data warehouse and a long-running sequence of user queries is called a *reader session.* The batch maintenance transaction is typically large and blocks the reader sessions. This renders the data warehouse

unavailable for the duration of the maintenance transaction. With the advent of the 24-hour, global internet market, the use of overnight maintenance runs will have to change and online processes developed. Incremental view maintenance which updates the data warehouse instantaneously in response to every change at the data source is expensive and gives rise to inconsistent results during the same reader session. An update from the data source will change the results a user might see over a sequence of queries. Quass and Widom (1997) discuss a possible approach to this problem by maintaining two versions of each tuple at the data warehouse simultaneously so that the reader sessions and the maintenance transactions do not block each other. A possible solution may need the integration with self maintenance techniques, where auxiliary views can be used to answer queries during maintenance transactions.

INDEXING SCHEMES

Indexing is the creation of access structures that provide faster access to the base data relevant to the restriction criteria of queries (Viguier and Datta 1997). The size of the index structure should be manageable so that benefits can be accrued by traversing such a structure. The traditional indexing strategies used in database systems do not work well in data warehousing environments. Most OLTP transactions typically access a small number of rows; most OLTP queries are *point* queries. B trees which are used in most common relational database systems are geared towards such *point* queries. They are well suited for accessing a small number of rows. An OLAP query typically accesses a large number of records for summarising information. For example, an OLTP transaction would typically query for a customer who booked on flight QFAN1234 on say April 25th; on the other hand an OLAP query would resemble a query such as *Return the number of customers who booked flights on QFAN1234 in April.* The second query would access more records and is typically a *range* query. B tree indexing scheme which is so apt for OLTP transactions is not the best suited to answer OLAP queries. There is a need to look at new indexing schemes and variants of existing schemes that will work well with OLAP queries. A number of indexing strategies have been suggested for data warehouses (Flanagan and Safdie 1997; Informix 1997; O'Neil and Quass 1997) in the literature. We review a few important strategies.

- **Value-List Index** - The value-list index consists of two parts. The first part is a balanced tree structure and the second part is a mapping scheme. The mapping scheme is attached to the leaf nodes of the tree structure and points to the tuples in the table being indexed. The tree is generally a B tree with varying percentages of utilisation. Oracle provides a B^* tree with 100% utilisation (Oracle 1992). Two different types of mapping schemes are in use. First one consists of a RowID list which is associated with each unique search-key value. This list is partitioned into a number of disk blocks chained together. The second scheme uses bitmaps. A bitmap is a vector of bits that store either a 0 or a 1 depending upon the value of a predicate. A bitmap B lists all rows with a given predicate P such that for each row r with ordinal number j that satisfies the predicate P, the jth bit in B is set. Bitmaps efficiently represent low-cardinality data, however to make this indexing scheme practical for high-cardinality data, compression techniques must be used. Value-list indexes have been shown in (O'Neil and Quass 1997) to outperform other access method in queries involving the MIN or MAX aggregate functions, as well as queries that compute percentile values of a given column.
- **Projection Index** - A projection index is equivalent to the column being indexed. If C is the column being indexed, then the projection index on C consists of a stored sequence of column values from C in the same order as the ordinal row number in the table from where the values are extracted. It has been shown in (O'Neil and Quass 1997) that projection indexes outperform other indexing schemes for performing queries that involve computation on two or more column values and appear to perform acceptably well in GROUP-BY like queries.
- **Bit-sliced Index** - A bit sliced index represents the key values of the column to be indexed as binary numbers and projects a set of *bitmap slices* which are orthogonal to the data held in the projection index. This index has been shown (O'Neil and Quass 1997) to particularly perform well for computing sums and averages. Also, it outperforms other indexing approaches for percentile queries if the underlying data is clustered, and for range queries whose range is large.
- **Data Index** - A DataIndex, like the projection index, exploits the positional indexing strategy (Viguier, Datta and Ramamritham 1997). The DataIndex avoids duplication of data by storing only the index and not the column being indexed. The dataindex can be of two specific types: Basic DataIndex and Join DataIndex (for more information, see (Viguier, Datta and Ramamritham 1997)).

PARALLELISM IN DATA WAREHOUSING

In this paper, we have discussed possible techniques to evaluate OLAP queries faster. All these techniques are restricted in their ways. The precomputation strategy needs to anticipate queries so that it can materialize them in the data warehouse. OLAP queries are of ad-hoc nature and restricts precomputing to the imagination of the

designer. The indexing schemes we have discussed provide faster access to the data stored in the warehouse, but does not reduce the size of tables stored at the data warehouse. One can say that the strategies presented provide faster query processing, but they need not be fast enough as perceived by the user. One mechanism that is recently being exploited by vendors to compute queries quickly is to execute the query in parallel by partitioning data among a set of processors. This can potentially achieve *linear speedup* and can significantly improve query response times. However to exploit this technology we need to address a few key issues such as parallel data placement and parallel join.

The existing data warehouse schema design strategies is not catered to parallel data placement, although the star schema suggests implicitly the need of partitioning the data into a set of dimension tables and fact tables. Many DBMS vendors claim to support parallel data warehousing to various degrees. Most of these products, however, do not use the dimensionality of data that exists in a data warehouse. The effective use of parallel processing in this environment can be achieved only if we are able to find innovative techniques for parallel data placement using the underlying properties of data in the warehouse. (Datta, Moon and Thomas 1998) use the data indexing strategy to provide efficient data partitioning and parallel resource utilisation. Effective algorithms that split the data among *n* parallel processors and perform parallel join operation have been illustrated.

DATA MINING ISSUES FOR DATA WAREHOUSES

Recent years have seen a continued growth in the size and number of databases and, more recently, in the development of data warehouses. Whilst much useful knowledge is contained implicitly in them, the large volumes of data involved have prevented their full exploitation. This has led to research into the automatic extraction or synthesis of knowledge from databases commonly termed data mining or knowledge discovery from databases.

In the late 1980s and early 1990s, data mining, and the wider field of knowledge discovery from databases, became a research area with roots in active and deductive databases, artificial intelligence research (specifically that research looking into machine learning), statistical analysis and information retrieval. More recently, the field has divided into two sub-areas:

- Autonomous and semi-autonomous knowledge discovery, which is looking at applying previous methods for
- the maintenance of summary databases, including a strong emphasis on the development of techniques that can incrementally keep summary databases and data warehouses up to date;
- the extraction of knowledge for the more highly structured data warehouses;
- the extraction of data from multi-dimensional databases including temporal and spatio-temporal databases.
- The development of databases that accommodate induction in databases and data warehouses.

From the standpoint of data warehouses, data mining has much to offer although to date, with a few notable exceptions, the 'mining' of data in commercial or industrial databases has largely been under the umbrella of OLAP technologies. Notable exceptions include (Fayyad, Weir and Djorgovski 1993; Keats and Loo 1997; Shortland and Scarfe 1994; Viveros, Nearhos and Rothman 1996; Ziarko, Golan and Edwards 1993) which show the utility of autonomous data mining. The predefined schemata (which may be in various formats as described earlier) provide a template from which mining can then proceed. As discussed in (Rainsford and Roddick 1999), learning from data warehouses has a number of advantages:

- The data is stored in a more or less structured manner. For example data is typically structured into (normalised or denormalised) relations that eliminate redundancy and can be joined in various ways to retrieve the required data sets from a database. In other database paradigms, either more or less structure is available. Nevertheless a certain amount of *apriori* known structure is available and can, and arguably should be, utilised in the mining process.
- In addition to the structure above, some domain knowledge may already be encoded implicitly or explicitly within the database. For example, the existence of a participation constraint may be flagged by a not null constraint. Similarly, the cardinality of relationships is also often explicit within database structure and constraints.
- High performance query, data manipulation and transaction tools are already available. This would include the database management system, associated query language, specialised hardware and other database tools. It therefore makes some sense to use these tools to interrogate the database where appropriate.

Similarly, the use of data resident in databases and data warehouses imposes a number of characteristics and constraints not necessarily encountered in smaller volume machine learning applications:

- The volume of data is typically large. For example, the SKICAT system has been developed to process three terabytes of graphic images resulting from a sky survey (Fayyad, Weir and Djorgovski 1993).

While this may be at the high end, data warehouse volumes are typically in or close to the Gigabyte range. Any data mining tool must therefore perform satisfactorily on large volumes of data.

- The data may contain noise. Data mining tools must provide adequate mechanisms for finding sufficiently accurate results from noisy data.
- The database may contain incomplete information. Not all information useful for the discovery of knowledge may actually be stored within the database. Likewise, much redundant and useless data may also be present. Data mining tools must therefore facilitate both the selection of relevant data, and learning with incomplete knowledge.
- In general, data is not collected for the purpose of knowledge discovery. As well as leading to some of the above problems this means that data may be formatted or interpreted inappropriately. Knowledge discovery tools must therefore be able to access data stored in various formats.

A number of models of the data mining process have been suggested. All split the mining process into a number of phases that are iterated as appropriate. The models suggested can be adapted to use with data warehouses as shown in the figure.

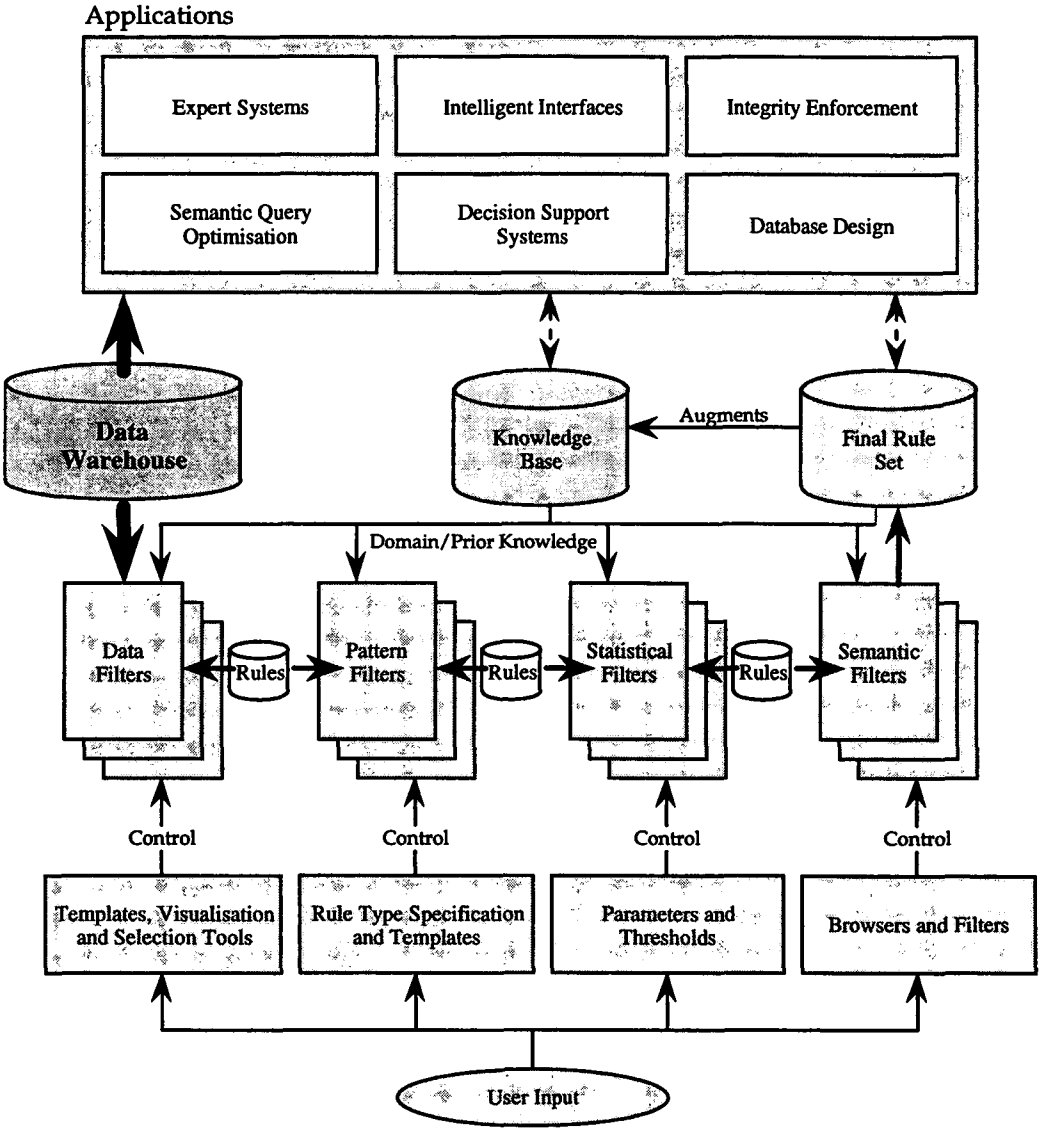

Figure 6. A Model of the Data Mining Process (from (Rainsford and Roddick 1999))

For any database, the number of possible rules that can be extracted is far greater than the number of facts in a data warehouse. Knowledge discovery can be viewed as a multi-stage process of selecting interesting rules from the total rule-space that exists within a database. This is therefore a process of progressively reducing the initial infinite rule space down to a small subset of useful rules. This reduction process is performed using a selection of filters that reduce the target rule space on the basis of source data, rule pattern, statistics and semantics. Special cases may exist where a phase of the filtering process does not exist within a system, and

therefore the filter will allow the rule space to pass unreduced. For example, semantic filtering may not be used if rules are to be used for query optimisation. Each of the filtering stages may therefore consist of zero or more filters, specified by the user or discovery system. The target rule set may be passed back and forth between the filters for reprocessing. A central controller coordinates the operation of the filters. As noted on the diagram the final rule set can be integrated into the existing knowledge base and both the knowledge base and the user may interact with each step of the rule-space reduction process. Note that the filtering processes may have a profound effect on the outcome of the data mining process. Thus the outcome of the data mining process is quasi-nondeterministic.

The rules derived from a data warehouse are clearly dependent on both the data held and the algorithm applied. Classification rules, for example, can be used to categorise examples into classes on the basis of known properties (Agrawal, *et al.* 1992), while a characteristic rule can be defined as *an assertion that characterises the concept satisfied by all of the relevant data in the database* (Han, Cai and Cercone 1993). Association rules provide a measure of correlation between the occurrence of items in set of transaction (Agrawal, Imielinski and Swami 1993). Other forms of rules, such as functional, causal and temporal can also be mined if the data warehouse data allows it; some of these are outlined in (Rainsford and Roddick 1999).

A number of surveys have been published that provide different perspectives on the development and use of data mining tools in various domains (Chen, Han and Yu 1996; Frawley, Piatetsky-Shapiro and Matheus 1992; Holsheimer and Siebes 1994) (Fayyad, Piatetsky-Shapiro and Smyth 1996; Fayyad, *et al.* 1996; Koperski, Adhikary and Han 1996; Rainsford and Roddick 1999). These and other publications indicate that a number of advanced utilities and approaches may be required in the future as follows:

Embedded Summarisation - Research into the construction of data cubes in databases (Gray, *et al.* 1997) suggests that OLAP applications would benefit from a number of advanced summarisation functions. Unfortunately, it is currently unclear how efficient query optimisation can be maintained if data cube operations are performed at the level of middleware. There is already some concern about whether current query optimisation techniques are able to accommodate data warehousing (Chaudhuri 1997).

Concept Hierarchies/Lattices - Many algorithms proposed for data mining rely on the idea of concept ascension. In addition, concept hierarchies would be a useful addition to many other application domains, including arguably, traditional transaction processing. The lack of accommodation of concept hierarchies has therefore necessitated more or less extensive, but always fairly similar, middleware layers to add concept hierarchies and/or lattices.

Sampling, clustering and approximation - While there may be specified bounds, many multimedia databases and real-time systems are commonly willing to accept *close answers now* in preference to *exact answers eventually*. For different reasons, data mining applications are also commonly content to accept approximate answers, perhaps based on the random sampling of data or of a given cluster or sub-set of the data.

Inductive queries - Data mining researchers and, to a lesser extent those involved with data warehousing, sometimes find that it is useful to relax the closed world assumption. That is, it is occasionally useful to ask questions of the database without necessarily assuming that either all the data is held or that anything not held in the database is untrue. Moreover, AI and deductive database research has long investigated, for example, the explicit holding of known falsities so that a tri-state logic of true, false and unknown can be returned. The development of other database paradigms to accommodate this may also be useful.

Incremental knowledge discovery - The volume of data currently being mined is outstripping the normal capacity of data mining systems to mine it, primarily due to the I/O-boundness of data mining routines and the fact that I/O channel speed improvements are not keeping pace with improvements in either storage capacity of CPU speed. It is thus important that both incremental discovery routines, which can amend data warehouse data in bursts as changes are made to the transaction database, and that restartable routines, which can be run whenever resources permit, such as overnight or during other periods of low activity, be developed. It would also be useful if cooperating data mining algorithms, each tailored to mine different aspects of the same data store, could be used. To enable both to happen, a common mechanism for storing interim results would be useful.

OTHER RESEARCH CHALLENGES AND CONCLUSIONS

The research challenges discussed so far are directly concerned with the problem of faster data integration. There are other challenges which are a result of the data warehousing concept. Here we mention a few such challenges which are currently active. One such problem arises due to the changing user requirements. As the

requirements of the user changes, the view definitions at the data warehouse change dynamically. There is a need to develop view adaptation techniques that do not need the entire recomputation of the materialized view at every change. In (Gupta, Mumick and Ross 1995; Mohania 1997; Mohania and Dong 1996), the authors have presented the view adaptation techniques for a data warehouse that recompute only parts of the view (if any) which cannot be derived from the existing materialized views. These techniques are applicable for SPJ queries. However, there is a need to devise adaptation algorithms for aggregated queries.

As discussed in (Munneke, Wahlstrom and Mohania 1999), the fragmentation of a multi-dimensional databases has not been widely researched. The fragmentation mechanisms employed to partition a database over multiple sites play an important role in the design of a distributed database system as proper fragmentation enhances query performance by defining appropriate units of distribution which allow for concurrent, parallel execution of queries. The parallel processing of OLAP operations to achieve quick response times is similarly important (Datta, Moon and Thomas 1998). This can be achieved by fragmenting multi-dimensional database into number of fragments. We identify two types of possible fragmentation. Firstly, *slice and dice* that selects subsets of data by `cutting up' the multi-dimensional hypercube representation of the global data into sub-cubes. *Slice and dice* is currently employed by some multi-dimensional databases that support fragmentation; for example the *Essbase* OLAP Server Partitioning option. Secondly, *severing* that divides the global data by removing dimensions from a hypercube. This method may be used to divide a cube into two cubes with different dimensions from each other. Operations performed during severing ensure that the original cube may be reconstructed from the two severed cubes (Munneke, Wahlstrom and Mohania 1999).

Another interesting problem is the problem of data expiry in the warehouse. Data that is materialized in the warehouse may not be needed after a point. There is a need to devise efficient schemes that determine the data expiry point and do effective garbage collection by removing the unnecessary tuples. The challenge that awaits the data warehousing community is the integration of Web data in the warehouse. The internet has become a repository of unstructured information and the need to incorporate such external information will require the development of effective schemes for extracting semi-structured and unstructured data for storing in the data warehouse.

CONCLUSIONS

A data warehouse is a central repository that contains highly aggregated, summarised and consolidated data, and is used for data analysis and decision support functions. To do effective data analysis, the quality of data is important which can be achieved by cleaning the data. This process involves data filtering, transformation and integration in order to get rid of data inconsistencies in the warehouse. In this paper, we have discussed several research issues in the areas of data warehouse modelling, view selection and maintenance, indexing and parallel query processing. Further, we have exploited the multi-dimensionality of data in the warehouse to introduce the concept of constraints on the data cube. We highlight the usefulness of the constraint set in every aspect of data warehousing, that is, design, implementation and maintenance. This paper also underlines the need for dynamic self-maintainability of materialized views and discusses the issues involved.

In this paper we have also discussed the advantages and differences in mining from data repositories such as data warehouses. It is our view that while the majority of commercially implemented mining systems are currently based around the interactive OLAP mode of knowledge discovery, the use of semi-autonomous mining search engines will become more commonplace once some research problems, such as rule useability, are resolved.

The area discussed in the paper also contains a number of research issues which will need to be dealt with before commercial information systems can leverage the maximum advantage from data warehouses and the associated technology. Apart from the technical issues discussed at various points in the paper, of particular immediate concern is the manner by which data warehouse design and implementation processes are seamlessly integrated with conventional information systems design methodologies.

Acknowledgments

We are grateful to Kirsten Wahlstrom for bringing an important omission to our attention.

REFERENCES

Agrawal, D., *et al.* (1997). "Efficient view maintenance in data warehouses". **ACM SIGMOD International Conference on Management of Data**, Tucson, Arizona, USA, pp. 417-427.

Agrawal, R., *et al.* (1992). "An Interval Classifier for Database Mining Applications". **Eighteenth International Conference on Very Large Data Bases**, Vancouver, Canada, pp. 560-573.

Agrawal, R., *et al.* (1997). **Modeling multidimensional databases**. Research Report Report IBM.

Agrawal, R., *et al.* (1993). "Mining Association Rules Between Sets of Items in Large Databases". **ACM SIGMOD International Conference on Management of Data**, Washington DC, USA, ACM Press. Vol. 22. pp. 207-216.

Bailey, J., *et al.* (1998). "Distributed view maintenance by incremental semijoin and tagging". **Distributed and Parallel Databases.** Vol. 6, No. 3, pp. 287-309.

Baralis, E., *et al.* (1996). "Conservative timestamp revised for materialized view maintenance in a data warehouse". **Workshop on Materialized Views**, Montreal, pp. 1-9.

Baralis, E., *et al.* (1997). "Materialized view selection in a multidimensional database". **International Conference on Very Large Databases**, Athens, Greece, Morgan Kaufmann. pp. 156-165.

Bauer, A. and Lehner, W. (1997). "The cube-query-language (CQL) for multidimensional statistical and scientific database systems". **Fifth International Conference on Database Systems for Advanced Applications (DASFAA)**, Melbourne, pp. 263-272.

Blakeley, J. A., *et al.* (1986). "Efficiently updating materialized views". **ACM SIGMOD International Conference on the Management of Data**, pp. 61-71.

Chaudhuri, S. (1997). "Panel on "Query Optimisation at the Crossroads"". **ACM SIGMOD International Conference on the Management of Data**, Tucson, Arizona, USA, pp. 509.

Chaudhuri, S. and Dayal, U. (1997). "An overview of data warehousing and OLAP technology". **SIGMOD Record.** Vol. 26, No. 1, pp. 65-74.

Chen, M.-S., *et al.* (1996). "Data mining: an overview from database perspective". **IEEE Transactions on Knowledge and Data Engineering.** Vol. 8, No. 6, pp. 866-883.

Datta, A., *et al.* (1998). **A case for parallelism in data warehousing and OLAP.** Department of MIS, University of Arizona, Tucson, AZ.

Dong, G. and Mohania, M. (1996). "Algorithms for view maintenance in mobile databases". **First Australian Workshop on Mobile Computing and Databases**, Monash University, Australia,

Fayyad, U., M., *et al.* (1996). "From Data Mining to Knowledge Discovery: An Overview". in **Advances in Knowledge Discovery and Data Mining.** AAAI Press/ MIT Press. pp. 1-34.

Fayyad, U., M., *et al.*, Ed. (1996). **Advances in Knowledge Discovery and Data Mining.** Menlo Park, California, AAAI Press.

Fayyad, U. M., *et al.* (1993). "Automated Analysis of a Large-Scale Sky Survey: The SKICAT System". **AAAI-93 Workshop on Knowledge Discovery in Databases (KDD '93)**, Washington D.C., AAAI Press, Menlo Park, California. pp. 1-13.

Flanagan, T. and Safdie, E. (1997). **Data Warehouse Technical Guide.** White Paper Report Sybase.

Frawley, W. J., *et al.* (1992). "Knowledge discovery in databases: an overview". **AI Magazine.** Vol. 13, No. 3, pp. 57-70. Also in Piatetsky-Shapiro, G. and Frawley, W.J., (eds.) 1991. *Knowledge discovery in databases.* AAAI Press/MIT Press, Menlo Park, CA. 1-30.

Gray, J., *et al.* (1997). "Data Cube: A Relational Aggregation Operator Generalising Group-by, Cross-Tab and Sub-Totals". **Data Mining and Knowledge Discovery.** Vol. 1, No. 1, pp. 29-54.

Griffin, T. and Libkin, L. (1995). "Incremental maintenance of views with duplicates". **ACM SIGMOD International Conference on the Management of Data**, San Jose, USA, pp. 328-339.

Gupta, A. and Mumick, I. S. (1995). "Maintenance of materialized views: problems, techniques and applications". **IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing.** Vol. 18, No. 2, pp. 3-18.

Gupta, A., *et al.* (1995). "Adapting materialized views after redefinitions". **ACM SIGMOD International Conference on Management of Data**, San Jose, USA, pp. 211-222.

Gupta, A., *et al.* (1993). "Maintaining views incrementally". **ACM SIGMOD International Conference on Management of Data**, pp. 157-166.

Han, J., *et al.* (1993). "Data-Driven Discovery of Quantitative Rules in Relational Databases". **IEEE Transactions on Knowledge and Data Engineering.** Vol. 5, No. 1, pp. 29-40.

Holsheimer, M. and Siebes, A. P. J. M. (1994). **Data mining: the search for knowledge in databases.** Technical Report Report CS-R9406. CWI, The Netherlands.

Hull, R. and Zhou, G. (1996). "A framework for supporting data integration using the materialized and virtual approaches". **ACM SIGMOD Conf. On Management of Data**, Montreal, Canada, pp. 481-492.

Huyn, N. (1997). "Multiple view self-maintenance in data warehousing environments". **International Conference on Very Large Databases**, Athens, Greece, pp. 26-35.

Informix (1997). **A New Generation of Decision-Support Indexing for Enterprise-wide Data Warehouses.** White Paper Report Informix.

Inmon, W. H. (1992). **Building the Data Warehouse.** QED Technical Publishing Group.

Keats, G. and Loo, S. (1997). "An intelligent business workbench for the insurance industry: using data mining to improve decision making and performance.". **Eighth International Database Workshop, Data Mining, Data Warehousing and Client/Server Databases.**, Hong Kong, Springer-Verlag Singapore. pp. 256-274.

Koperski, K., *et al.* (1996). "Knowledge Discovery in Spatial Databases: Progress and Challenges". **ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery**, Montreal, Canada, pp. 55-70.

Kuchenhoff, V. (1991). "On the efficient computation of the difference between consecutive database states". in **Second International Conference on Deductive Object-Oriented Databases**. Lecture Notes in Computer Science. Vol. 566. Springer-Verlag. pp. 478-502.

Labio, W. J., *et al.* (1997). "Physical database design for data warehouses". **Thirteenth International Conference on Data Engineering**, Birmingham, U.K., IEEE Computer Society. pp. 277-288.

Lehner, W., *et al.* (1996). "Cross-DB: A feature-extended multidimensional data model for statistical and scientific databases". **International Conference on Information and Knowledge Management**, Rockville, Maryland, USA, pp. 253-260.

Li, C. and Wang, X. S. (1996). "A data model for supporting on-line analytical processing". **International Conference on Information and Knowledge Management**, Rockville, Maryland, USA, pp. 81-88.

Mohania, M. (1997). "Avoiding re-computation: View adaptation in data warehouses". **Eighth International Database Workshop**, Hong Kong, pp. 151-165.

Mohania, M. and Dong, G. (1996). "Algorithms for adapting materialized views in data warehouses". **International Symposium on Cooperative Database Systems for Advanced Applications**, Kyoto, Japan, pp. 62-69.

Mohania, M., *et al.* (1997). "Incremental maintenance of materialized views". **Eighth International Conference on Database and Expert Systems Applications (DEXA '97)**, Toulouse, France, Springer-Verlag. pp. 551-560.

Munneke, D., *et al.* (1999). "Fragmentation of Multidimensional Databases". **Tenth Australasian Database Conference**, Auckland, New Zealand, Springer. pp. 153-164.

O'Neil, P. and Quass, D. (1997). "Improved query performance with variant indexes". **ACM SIGMOD International Conference on Management of Data**, Tucson, Arizona, USA, pp. 38-49.

Oracle (1992). **Oracle 7 Server Concepts Manual**. Redwood City, CA, Oracle Corporation.

Quass, D., *et al.* (1996). "Making views self-maintainable for data warehousing". **International Conference on Parallel and Database Information Systems**, Miami Beach, Florida, USA,

Quass, D. and Widom, J. (1997). "On-line warehouse view maintenance for batch updates". **ACM SIGMOD International Conference on Management of Data**, Tucson, Arizona, USA, pp. 393-404.

Rainsford, C. P. and Roddick, J. F. (1999). "Database Issues in Knowledge Discovery and Data Mining". **Australian Journal of Information Systems.** Vol. 6, No. 2, pp. 101-128.

Ramakrishnan, R., *et al.* (1994). "Efficient incremental evaluation of queries with aggregation". **International Logic Programming Symposium**,

Ross, K. A., *et al.* (1996). "Materialized view maintenance and integrity constraint checking: Trading space for time". **ACM SIGMOD International Conference on Management of Data**, Montreal, Canada,

Roussopoulos, N. (1982). "View indexing in relational databases". **ACM Transactions on Database Systems.** Vol. 7, No. 2, pp. 258-290.

Segev, A. and Fang, W. (1990). "Currency based updates to distributed materialised views". **IEEE International Conference on Data Engineering**, Los Angeles, California, pp. 512-520.

Segev, A. and Park, J. (1989). "Maintaining materialised views in distributed databases". **IEEE International Conference on Data Engineering**, Los Angeles, California, pp. 262-270.

Shortland, R. and Scarfe, R. (1994). "Data mining applications in BT". **BT Technology Journal.** Vol. 12, No. 4, pp. 17-22.

Viguier, I. R. and Datta, A. (1997). **Sizing access structures for data warehouses**. Technical report Report Dept. of MIS, University of Arizona, Tucson, AZ.

Viguier, I. R., *et al.* (1997). **'Have your data and index it, too', efficient storage and indexing for data warehouses**. Technical Report Report GOOD-TR-9702. Department of MIS, University of Arizona.

Viveros, M. S., *et al.* (1996). "Applying Data Mining Techniques to a Health Insurance Information System". **Twenty-second International Conference on Very Large Data Bases (VLDB '96)**, Mumbai, India, Morgan Kaufmann Publishers, Inc. San Francisco, USA. pp. 286-293.

Widom, J. (1995). "Research Problems in Data Warehousing". **Fourth International Conference on Information and Knowledge Management (CIKM)**, Baltimore, Maryland, pp. 25-30.

Yang, J., *et al.* (1996). **A framework for designing materialized views in data warehousing environment**. Techn ical Report Report HKUST-CS96-35. Department of Computer Science, The Hong Kong University of Science and Technology.

Zhuge, Y., *et al.* (1995). "View maintenance in a warehousing environment". **ACM SIGMOD International Conference on Management of Data**, San Jose, USA, pp. 316-327.

Zhuge, Y., *et al.* (1996). "The strobe algorithms for multi-source warehouse consistency". **International Conference on Parallel and Distributed Information Systems**, Miami Beach, Florida,

Ziarko, W., *et al.* (1993). "An Application of Datalogic/R Knowledge Discovery Tool to Identify Strong Predictive Rules in Stock Market Data". **AAAI-93 Workshop on Knowledge Discovery in Databases (KDD '93)**, Washington D.C., AAAI Press, Menlo Park, California. pp. 89-101.