Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 9 (3): 401-414, 2009
ISSN 1812-5654

Materialized View Selection in Data Warehousing: A Survey

C.A. Dhote and M.S. Ali
Department of Computer Science and Engineering, PRM,
Institute of Technology and Research, Badnera, Amravati, India

Abstract: This study presents the critical survey of the methodologies to select materialized view in more efficient way. In this study, we are summarizing all these methodologies with critical analysis. Advanced solutions are particularly focusing the evolutionary optimization methods. We have analyzed and compartmentalized the available literature on the basis of relevant evaluation parameters. Important books, Ph.D thesis, links, etc. are also given in study. To work out this study we have gone through more than fifty research papers. This study may be helpful to the researchers, who are working in the domain of the Data Warehouse focusing on the materialized view selection.

Key words: Data warehouse, materialized view, query process plan, local search, genetic algorithm

INTRODUCTION

A set of significant new concepts and tools have evolved in to a new technology that makes it possible to attack the problem of providing all the key people in the enterprise with access to whatever level of information needed for the enterprise to survive and prosper in an increasingly competitive world. The term that has come to characterize this new technology is Data Warehousing. Data warehousing is an emerging approach for effective decision support. According to Inmon (1996), a data warehouse is a subject-oriented, integrated, time-varying, nonvolatile collection of data that is used primarily in organizational decision making. The need for data warehousing techniques is justified due to decision support queries, which are ad-hoc user queries in various business applications. In these applications, current and historical data are comprehensively analyzed and explored, identifying useful trends and creating summaries of the data, in order to support high-level decision making in a data warehousing environment. A class of queries typically involves group-by and aggregation operators.

Maintaining or updating a warehouse is not cheap because propagating updates from source data to the aggregates in the warehouse is a time consuming process that usually must be done off-line. Many issues should be studied, including how to detect changes and how to develop algorithms for warehouse maintenance that are able to balance update cost against query response time.

From a computer science perspective, a data warehouse is a collection of materialized views derived from base relations that may not reside at the warehouse. Therefore, a data warehouse is considered as a definer and storage of views. When a view is defined, the database system stores the definition of the view itself, rather than the result of evaluation of the relational algebra expression that defines the view. Hence, a view is a derived relation defined in terms of base relations. A view thus defines a function from a set of base tables to a derived table; this function is typically recomputed every time the view is referenced. According to the perspective of materialized views, at the abstract level the contents of the data warehouse are regarded as a set of materialized views defined over the data sources. These materialized views are designed based on the user's requirements (e.g., frequently asked queries). The benefit of using materialized views is significant. Since index structures can be built on materialized views, consequently, database access to the materialized view can be much faster than recomputing the views. A materialized view is just a cache, which is a copy of the data that can be accessed quickly. Integrity checking and query optimization can also benefit from materialized views. In short, when a view is defined, normally the database stores only the query defining the view. In contrast, a materialized view is a view whose contents are computed and stored. It is cheaper in many cases to read the contents of a materialized view than to compute the contents of the view by executing the query defining the view. Materialized views are important for improving performance in some applications.

Though some review papers are available, but most of them are from year 1980 to 1990. In last decade, many researchers have proposed the solution strategies for

Corresponding Author: Chandrashekhar A. Dhote, Department of Computer Science and Engineering, PRM, Institute of Technology and Research, Badnera, Amravati, India

materialized view selection problem. But year 2000 onwards, some advance techniques are used to find the solution e.g. simulated annealing, genetic algorithm etc. This study is intended for the beginners in the domain of materialized view selection problem. This study provide the details of basic data structures, mathematics, cost modeling, books, Ph.D thesis, links, glossary of key terms, benchmark database and critical analysis on past and present methods.

MATERIALIZED VIEW SELECTION PROBLEM AND COST MODEL

In materialized view selection problem, the main objective is either the minimization of a cost function or a constraint. A constraint can be user oriented or system oriented. Attempting to satisfy the constraints can result in no feasible solution to a view selection problem. Most of the approaches comprise in their design the minimization of a cost function. Many view selection problems define by Gupta (1997), Yang *et al.* (1997) and Baralis *et al.* (1997) and take as input the queries that the Data Warehouse has to satisfy for an initial or an incremental design. The overall query evaluation cost is the sum of the cost of evaluating each input query rewritten over the materialized views. This sum can also be weighted, each weight indicating the frequency, or importance of the associated query. The approaches aim at minimizing the query evaluation cost (Harinarayan *et al.*, 1996; Gupta and Mumick, 1996; Shukla *et al.*, 1998). The materialized views are maintained using an incremental approach. In an incremental approach, only the changes that must be applied to the view are computed using the changes of the source relations (Blaiseley *et al.*, 1986; Griffin and Libkin, 1995; Quass, 1996). These view changes are then applied to the materialized view. The view maintenance cost is the sum of the cost of propagating each source relation change to the materialized views. This sum can be weighted, each weight indicating the frequency of propagation of the changes of the associated source relation. The expressions used to compute the changes to be applied to a view involve the changes of the source relations and are called maintenance expressions. When the source relation changes affect more than one materialized view, multiple maintenance expressions need to be evaluated. The techniques of multi query optimization can be used to detect common sub expressions between maintenance expressions in order to derive an efficient global evaluation plan for these maintenance expressions (Sellis, 1988; Shim *et al.*, 1994). Maintaining the query evaluation cost and the view maintenance cost are conflicting requirements. Low view maintenance cost can be achieved by replicating source relations at the Data Warehouse; in this case the query evaluation cost is high. Low query evaluation cost can be obtained by materializing at the Data Warehouse all the input queries. In this case the view maintenance cost will be high. For this reason, one can choose a linear combination of the query evaluation and view maintenance cost. In most of the research related to the materialized view selection used the linear cost model (Harinarayan *et al.*, 1996), which is used for view cost evaluation. Only difference occurs at the assumptions which are used in the evaluation of mathematical model of the cost. Analytical justification of linear cost model based on graph theory is given in (Dhote and Ali, 2007). General linear cost model is described as follows:

Let us assume that a set of queries $Q = (Q_1, Q_2,..., Q_m)$ are defined over a set of source relations $S = (S_1, S_2,..., S_n)$ and a multi query graph G. Let GQ_i be the query DAG for Q_i, i = 1, ..., m in G. $E(GQ_i)$ denotes the cost of evaluating Q_i, using GQ_i. The query evaluation cost of G is:

$$E(G) = \sum_{i=1}^{m} fQ_i E(GQ_i) \quad (1)$$

where, fQ_i is query frequency.

Let GS_i where i = 1, ..., n be the change propagation DAGs for Q_i i = 1, ..., m in G. $M(GS_i)$ denotes the cost of propagating the changes of S_i to the materialized views using GS_i. The view maintenance cost of G is:

$$M(G) = \sum_{i=1}^{n} fS_i M(GS_i) \quad (2)$$

where, fS_i is source relation updation frequency. The values of query frequency and base relation updation frequency can be assumed for the experimentation.

In literature it is found that most of the researchers used this cost model for the evaluation of the total cost of all processing. Particularly, both the cost functions are considered as the independent function and hence the problem can be of multiobjective optimization. In some cases, both these functions are considered together as the single objective function, so the problem can be of single objective optimization.

View selection problem can be described as: How to select an appropriate set of materialized views from a certain graph G, so that the total query processing cost for the supported queries and the total maintenance cost of these materialized views is minimal.

Given a G, let M be a set of views in a G to be materialized, fq, fs the frequency of executing queries and frequency of updating base relations, respectively. Furthermore for each v∈M, let E (Gq_i (v)) and M (Gs_i (v)) denote the cost of access for query Q using v and the cost of maintenance of view v base on changes to base relation s, respectively (where, v∈QN is the set of queries and s∈SRN is the set of base relations). Then the query processing cost will be:

$$E(G(v)) = \sum_{i=1}^{m} fQ_i E(GQ_i(v)) \quad (3)$$

And the materialized view maintenance cost will be:

$$M(G(v)) = \sum_{i=1}^{n} fS_i M(GS_i(v)) \quad (4)$$

Thus the total cost of materializing a view v is:

$$\text{Total cost } (v) = E(G(v)) + M(G(v)) \quad (5)$$

Therefore, the total cost of materializing a set of views M is Total cost:

$$\text{Total cost} = \sum_{v \in M} \text{Total cost}(v) \quad (6)$$

Equation (3) and (4) can be use for multi objective optimization and Eq. 5 can be use for single objective optimization.

MATHEMATICAL BACKGROUND

To solve the view selection problem, mathematical formulation is the first step. In view selection problem mathematical formulation, data structures are required to represent the view selection. For this the concept of discrete mathematics and graph theory like set theory, graph theory, lattices are generally used. In following sub sections the fundamentals of the above mentioned concepts are discussed with the help of example.

Relational algebra: Relational algebra falls in to the category of procedural query language. Queries in relational algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result. This property makes it easy to compose operators to form a complex query- a relational algebra expression is recursively defined to be a relation, a unary algebra operator applied to a single expression, or a binary algebra operator applied to two expressions.

Some of the basic relational algebra operators which are involved in the query processing and their occurrence play an important role in the query optimization process and further, in to the query evaluation plan. The fundamental operators of the relational algebra are Selection (σ), Projection (Π), Union (∪), Intersection (∩), Cross-product or Cartesian product (×), Set Difference (-) and Join (∞).

Directed acyclic graph: A directed acyclic graph, also called a dag or DAG, is a directed graph with no directed cycles that is, for any vertex v, there is no nonempty directed path that starts and ends on v. DAGs appear in models where it doesn't make sense for a vertex to have a path to itself; for example, if an edge u< = v indicates that v is a part of u, such a path would indicate that u is a part of itself, which is impossible. Directed acyclic graphs can be used to represent a number of interesting relations. This includes trees, but is less general than class of all directed graphs.

AND/OR graph: Many complex problems can be broken down into a series of sub problems such that the solution of all or some of these results in the solution of the original problem. These sub problems may be broken down further in to sub-sub problems and so on until the only the problems remaining are sufficiently primitive as to be trivially solvable. These breaking down of the complex problem in to several sub problems can be represented by a directed graph like structure in which nodes represent problems and descendents of a node represent the sub problems associated with it. AND graph is represented as in Fig. 1, a problem A that can be solved by either solving both the sub problems B and C or the single sub problems D or E.

Groups of sub problems that must be solved in order to imply a solution to the parent node are joined together by an arc going across the respective edges i.e. the arc

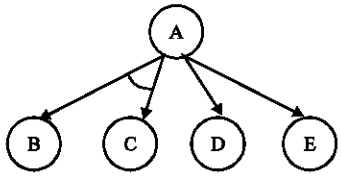

Fig. 1: AND graph

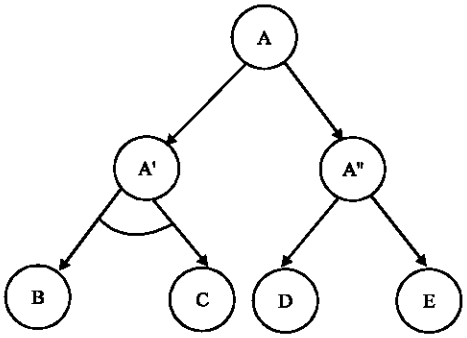

Fig. 2: Graphs representing AND/OR problem

across the edges <A, B> and <A, C>. By introducing dummy nodes as in Fig. 2, all nodes can be made to be such that their solution requires either all descendents to be solved or only one descendent to be solved. Node of first type are called AND nodes and those of the later type OR nodes. Nodes A and A of Fig. 2 are OR nodes while node A is an AND node. AND nodes will be drawn with an arc across all edges leaving the node. Nodes with no descendents are termed terminal.

Lattices: A data cube allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts. The interesting property of cubes is that the n-D data can be represented as a series of (n-1)-D cubes. Decision support applications involve complex queries on very large databases. Since response time should besmall, query optimization is critical. Users typically view the data as a multidimensional data cubes. The $\subset$ operator imposes a partial ordering on the queries. This is related to the views of a data cube problem forming a lattice. In order to be a lattice, any two elements (views or queries) must have at least upper bound and a greatest lower bound according to the $\subset$ ordering. However, in practice, one only needs the assumption that $\subset$ is a partial order and that there is a top element, a view upon which every view is dependent. Consider two queries Q_1 and Q_2. $Q_1 \subset Q_2$ can be defined if Q_1 can be answered using only the results of Q_2. It is then said that Q_1 is dependent on Q_2. In most of the applications, dimensions of a data cube consist of more than one attribute and the dimensions are organized as hierarchies of these attributes. A simple example shown in Fig. 3 is that of organizing the time dimension in to the hierarchy: day, month and year.

In the presence of hierarchies, the dependency lattice is more complex than a hypercube lattice. Hierarchies introduce query dependencies that one must accounts for when determining what queries to materialize.

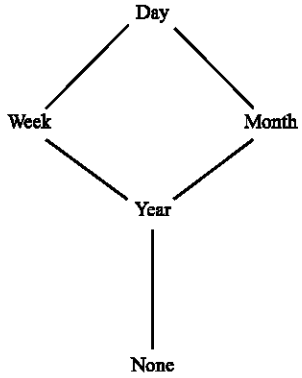

Fig. 3: Lattice example

REVIEW OF EXISTING WORK

The data warehouse problem through materialized views is usually stated as the view selection problem. When designing a data warehouse, it is extremely important to minimize the cost of answering queries because the warehouse is very large, queries are often ad hoc and complex and decision support application requires short response time. The determination of the optimal collection of views for available storage space and minimum query cost is referred to as the view selection problem. This view selection problem is totally different from the view selection problem under the disk space constraint. With numerous numbers of the base tables (with schemas in hundreds attributes) from dozens of data sources, it would be very challenging to decide which views should be materialized.

View selection problem can also be solved under the different types of constraints. For example, space constraints, time constraints, aggregation and grouping constraints, source availability constraints and currency constraints etc. Roussopoulos (1982), Gupta (1997), Theodoratos and Bouzeghoub (1999), Gupta *et al.* (1997) and Liang *et al.* (2001) explored the problem of selecting a set of materialized views for answering queries under the presence of updates and a global space constraint. When aggregation and grouping are present, view selection algorithms can be handled by constructing view graph structures. The view graphs express how source data is materialized into views. The algorithm basically creates the search space of possible configurations of materialized views. Ad hoc rules are used to limit the size of the generated 3-D view graphs. An important research issue is how to incorporate a pruning strategy for the algorithm with specialized view selection algorithms.

Appropriate data structures are needed for view selection. Design problems using views typically selects a set of views to materialize in order to optimize the query evaluation cost, view maintenance cost, or both, with certain constraints. For example, greedy algorithms have been provided for queries represented as AND/OR graph (Gupta, 1997). Gupta (1997) present a theoretical framework for the general view selection problem and polynomial-time algorithms for some special cases, which lower bound the benefit of the optimal solution. In particular, both query response time and the maintenance cost is to be minimized for a bounded space. Gupta (1999) extends the study in Gupta (1997) to address the problem of selecting views to materialize under the constraint of a given amount of total maintenance time. The literature is full of instances where researchers have used different data structures to solve the View Selection Problem in Data Warehouses and proposed different methodologies. This section provides a brief review of some of the approaches that have been discussed in the literature.

Harinarayan *et al.* (1996) model the view selection problem for the data cube using a lattice framework. The data cube lattice is a graph-representation of the CUBE operator that captures the dependencies among the different elements; views. Using this framework a greedy algorithm is used to pick a set of views with the maximum benefit. The benefit of a solution is defined as the total reduction in query response time compared with an empty set of views. A very important factor for the view selection problem is the constraints used for selection. For example, Theodoratos and Bouzeghoub (1999) refined the view selection problem to accommodate both source availability constraints (which are concerned with frequency of a data source accessed for view maintenance) and currency constraints (which are concerned with the aging of old data elements in the data warehouse). If constraints were not satisfactory, the source relation which caused the constraint violation would be identified. Otherwise, an algorithm was used to compute the minimal update frequencies to achieve the desired data currency. This work also concerned data quality in warehousing environment. Gupta *et al.* (1997), investigated the combined view and index selection problem under a given space constraint. Since the materialized views are large, having them pre-computed and stored within the data warehouse using relational tables is in many cases wasteful, unless these are indexed in order to support fast access to individual records. In present study presented a family of algorithms of increasing time complexity that consider also indices (B- trees) for the selected views.

A method of decomposing the data cube into a hierarchy of view elements that correspond to partial and residual aggregations works with an algorithm that picks a non-redundant set of such elements and minimizes query response time (Smith *et al.*, 1998). If extra space is available, a second algorithm that releases the non-redundancy requirement and focuses on selecting a set of elements that minimizes processing cost for target storage bound. The main drawback of this approach is the extremely high complexity of the decomposition step, which makes the algorithms impractical even for a limited number of dimensions/views. Query performance is tremendously improved as more views are materialized. With the ratio of cost to disk volume is constantly dropping, disk storage constraint is now no longer the limiting parameter in the view selection but the time to refresh the materialized set during updates. Liang *et al.* (2001) explained the view selection problem under the maintenance time constraint with two heuristics algorithms. The key underlying these algorithms is to define good heuristic functions and to reduce the problem to some well-solved optimization problems. Baralis *et al.* (1997) and Yang *et al.* (1997) presented various algorithms for minimizing the response time and the maintenance overhead without any resource constraint. Yang *et al.* (1997) also presented a framework for analyzing the issues in selecting views to materialize which lead to heuristic algorithm that provide a feasible solution based on individual optimal query plans. Labio *et al.* (1997) used an A* search to pick the best set of views when only the maintenance cost is to be minimized.

There are many other approaches on selection of common views. A data warehouse may contain multiple views which can be defined over overlapping portions of data. Data warehouses may contain multiple views with different query frequencies. When these views are related to each other and defined over overlapping portions of the base data, then it may be more efficient to materialize the certain shared portions of the base data from which the warehouse views can be derived. Identifying the shared views or shared parts of a particular view can therefore improve data warehouse performance by diminishing the number of views which have to be materialized, as well as reducing the amount of data requiring frequent refreshment. Theodoratos *et al.* (1999) defined a generic method that, given a set of SPJ queries to be satisfied by the data warehouse, generates all the significant sets of materialized views that satisfy all the input queries. This process is complex since common sub expressions between the queries need to be detected and exploited. In addition, algorithms have been developed so that a materialized view set selected in this way fits in the

space allocated to the data warehouse for materialization and minimizes the combined overall query evaluation and view maintenance cost. Zhang *et al.* (1999) explored the use of a genetic algorithm for the selection of materialized views based on multiple global processing plans for many queries.

The approach based on hypercube lattice is explained Harinarayan *et al.* (1996). The most common case of the hypercube lattice is considered and examined the choice of materialized views for hypercube in detail, giving some good tradeoffs between the space used and the average time to answer a query. In this research the problem of deciding which set of cells (views) in the data cube to materialize in order to minimize the query response time is investigated. Materialization of views is an essential query optimization strategy for decision support applications. Right selection of the views to materialize is critical to the success of the strategy (Harinarayan *et al.*, 1996). Shukla *et al.* (1998) proposed a modified faster algorithm that under some assumptions achieves the same benefit bound. A theoretical framework for the data warehouse configuration problem in terms of the relational model is proposed (Theodoratos and Sellis, 1997). A method for dealing with the problem was developed by formulating it as a state space optimization problem and then solving it using an exhaustive incremental algorithm as well as a heuristics algorithm. A case was considered where auxiliary views are stored in the Data Warehouse solely for reducing the view maintenance cost. In this study it was considered that there are no space restrictions in the Data warehouse and space is not the problem.

The approach is focused on the experimental evaluation of an exhaustive algorithm and develops greedy and heuristic algorithms that expand only a small fraction of the states produced by the exhaustive algorithm (Ligoudistianos *et al.*, 1998). The algorithms are described in terms of a state space search problem. The data warehouse configuration problem is formulated as a state space search problem based on a representation of views and queries using conjunctions of selection and join atomic predicates. A realistic cost model for query processing and view maintenance has been developed. Two algorithms: r-greedy algorithms that prune the state space and compare their performance using various criteria. In addition, heuristic algorithm that searches a small fraction of the state space and reports a sub-optimal solution, which is based on the greedy algorithms and compared its performance with the r-greedy algorithms, is developed. Mohania *et al.* (1999) discussed the problem of incremental maintenance of materialized views in data warehouses. Views defined by relational algebraic operators and aggregate functions were considered. It is shown that a materialized view can be maintained without accessing the view itself by materializing and maintaining additional relations. These relations are derived from the intermediate results of the view computation. An algorithm for determining what additional relations need to be materialized in order to maintain a materialized view incrementally was proposed. Further an efficient incremental algorithm for updating the materialized view (and the additional relations) based on the optimized operator tree used for evaluating the view as a query is proposed. One important feature of the algorithm is that it derives the exact change to every materialized additional relation, including the materialized view, without accessing the view itself. This feature is important to ensure the correctness of the update to views defined by aggregate functions. Assumption made was; views that contain relational algebraic operators and aggregate operators and can be represented by operator trees (Korth and Silberschatz, 1986).

It was assumed that duplicates are not retained in the materialized views. It was shown that a materialized view can be maintained efficiently by maintaining and materializing some additional results at the warehouse, called auxiliary relations, which may or may not contain intermediate results of the view. While deriving the auxiliary relations, referential integrity constraints are generated between the auxiliary relations and base relations. An algorithm for determining which auxiliary relations are needed in order to maintain a view is given. These relations make it possible to maintain an aggregate view without recomputing the intermediate results from scratch, thus significantly reducing the total computing and communication cost. In addition to reducing the cost of maintaining a view, storing auxiliary relations has additional benefits. Firstly, these auxiliary relations can be used in maintaining multiple views having common sub expressions, where the auxiliary relations will correspond to these sub expressions. Secondly, they may also be used for answering ad-hoc queries in data marts, where each data mart contains a subset of data in the data warehouse relevant to a particular domain of analysis (Chaudhuri and Dayal, 1997; Quass *et al.*, 1996). Lastly, the relations may be used to maintain a view when the view definition itself is slightly modified (Mohania, 1997).

To find the solution to the view selection problem, an evolutionary approach is described (Horng *et al.*, 1999). Genetic Local Search (GLS) algorithm (Ishibuchi *et al.*, 1997; Kolen and Pesch, 1994; Merz and Freisleben, 1997) is a hybrid heuristic that combines the advantages of population-based search such as Genetic algorithm and local optimization. Local search iteratively moves from one

solution to a better one in its neighborhood until a local minimum is reached (Davis and Smith, 1987; Goldberg, 1989). While it quickly finds good solutions in small regions of the search space, the genetic operators are suitable for exploring the whole search space in order to identify interesting regions.

The issue of designing a DW, in the context of the relational model, by selecting a set of views to materialize in the DW is discussed (Theodoratos and Sellis, 1999). A theoretical framework for the DW design problem, which concerns the selection of a set of views that (a) fit in the space allocated to the DW, (b) answer all the queries of interest and (c) minimize the total query evaluation and view maintenance cost. The problem is formulated as a state space search problem by taking into account multiquery optimization over the maintenance queries (i.e. queries that compute changes to the materialized views) and the use of auxiliary views for reducing the view maintenance cost. Finally, incremental algorithms and heuristics for pruning the search space are presented. Use of an evolutionary algorithm for materialized view selection based on multiple global processing plans for queries (Zhang *et al*., 2001). A hybrid evolutionary algorithm is applied to solve three related problems. The first is to optimize queries. The second is to choose the best global processing plan from multiple global processing plans. The third is to select materialized views from a given global processing plan.

Evolutionary algorithms use a randomized search strategy similar to biological evolution for good solutions. Although an evolutionary algorithm resembles randomized algorithms in this aspect, the approach shows enough differences to warrant a consideration of its own. The basic idea is to start with a random initial population and generate offspring by random variations (e.g., crossover and mutation). The fittest members of the population survive the subsequent selection; the next generation is based on these. The algorithm terminates as soon as there is no further improvement over a period or after a predetermined number of generations.

A randomized approach for incrementally selecting a set of views that are able to answer a set of input user queries locally while minimizing a combination of the query evaluation and view maintenance cost is developed (Theodoratos *et al*., 2001). In this process common sub-expressions among new queries and between new queries and old views have been exploited. The approach is based on the Simulated Annealing process.

To find an efficient plan for the maintenance of a set of materialized views, by exploiting common sub expressions between different view maintenance expressions is presented by Mistry *et al*. (2001). In particular, it has been shown how to efficiently select (a) expressions and indices that can be effectively shared, by transient materialization, (b) additional expressions and indices for permanent materialization and (c) the best maintenance plan-incremental or recomputation-for each view. These three decisions are highly interdependent and the choice of one affects the choice of the others. A framework was developed that cleanly integrates the various choices in a systematic and efficient manner.

Lee and Hammer (2001) focused on an efficient solution to the maintenance-cost view selection problem using a genetic algorithm for computing a near-optimal set of views. Specifically, the view selection problem in the context of OR view graphs has been explored. In this study, a solution to the maintenance cost view selection problem which minimizes query response time given varying upper bounds on the maintenance cost, assuming unlimited amount of storage space because storage space is regarded cheap and not a critical resource is focused. Specifically the view selection problem in the context of OR view graphs, in which any view can be computed from any of its related views has been explored.

A scalable algorithm for determining whether part or all of a query can be computed from materialized views and describes how it can be incorporated in transformation-based optimizers is presented (Goldstein and Larson, 2001). The main contributions of this paper are (a) an efficient view matching algorithm for views composed of selections, joins and a final group-by (SPJG views) and (b) a novel index structure (on view definitions, not view data) that quickly narrows the search to a small set of candidate views on which view-matching is applied. The version of the algorithm described here is limited to SPJG views and produces single-view substitutes.

Due to the space constraint and maintenance cost constraint, the materialization of all views is not possible. Therefore, a subset of views needs to be selected to be materialized. The problem noticed is NP-hard, therefore, exhaustive search is infeasible. A View Relevance Driven Selection (VRDS) algorithm based on view relevance to select view is developed (Valluri *et al*., 2002).

The query processing cost and the view maintenance cost was taken into consideration. In this paper, the concept of view relevance is introduced. After having obtained the optimal AND DAG, one wishes to select a subset of the views to materialize. This is based on the view relevance, which indicates how the presence of a view in the set, affects the benefit of the other views, thus affecting the total query processing cost and update maintenance cost. For implementing the View Selection

algorithm, an array is used to indicate whether a node is visited or not. It is an array of size equal to the number of nodes. All the elements of the array are initially assigned a value 0. Whenever a node is considered for materialization, its value is changed to 1. The VRDS algorithm strikes a balance between the query processing cost and the view maintenance cost, whereas greedy algorithm is focused mainly on updates and MVPP algorithm on selecting all beneficial views.

A constrained evolutionary algorithm is proposed (Yu *et al*., 2003). Constraints are incorporated into the algorithm through a stochastic ranking procedure where no penalty functions are used and constraint handling technique, i.e., stochastic ranking, can deal with constraints effectively. The algorithm proposed is able to find a near-optimal feasible solution and scales with the problem size well. First, pools of bit string genomes are generated randomly. This is the initial population. Each genome represents a candidate solution to the problem to be solved. The length of this genome is the total number of vertices in the lattice; 1 and 0 mean that the vertices need to be materialized or not, respectively.

Based on whether the current materialized views will be used in computing the new views and whether the data warehouse will query the remote data sources for additional data to do the computation, the data warehouse view maintenance techniques are classified into four major categories: self-maintainable recomputation, not self-maintainable recomputation, self-maintainable incremental maintenance and not self-maintainable incremental maintenance.

The approach provides a comprehensive comparison of the techniques in these four categories in terms of the data warehouse space usage and number of rows accessed in order to propagate an update from a remote data source to a target materialized view in the data warehouse (Wang *et al*., 2004). The analysis shows that self-maintainable incremental maintenance performs the best in terms of both space usage and number of rows accessed. The comparison of these categories is given in Table 1. Both the not self-maintainable recomputation and not self-maintainable incremental maintenance approaches suffer from some common disadvantages. As the remote data sources have to process queries from the data warehouse that consume their limited local resources, the OLTP system will be slow. Once a data source is unavailable, the data source will not be able to answer queries sent from the data warehouse in time. It will block the data warehouse view maintenance process. The not self-maintainable incremental maintenance approach has some additional disadvantages. To avoid the anomaly problem, the view maintenance process must be designed carefully.

Among all the four categories, self-maintainable incremental maintenance (Quass *et al*., 1996; Cui and Widom, 1999; Hull and Zhou, 1996) is the best in terms of

Table 1: Comparison of four categories

Category	Advantage	Disadvantage
Self-Maintainable Recomputation	Data warehouse view maintenance operations are totally separated from OLTP operations. Unavailable source will not block the data warehouse view maintenance process.	Data are replicated at data warehouse. Need extra data storage for replicate data. Have to implement and maintain data transfer processes to transfer data from. Sources to data warehouse.
Not Self- Maintainable Recomputation	Very simple to implement. No replicate data at the data warehouse. No extra data storage for replicate data. Do not have to implement and maintain data transfer processes to transfer data from sources to data warehouse.	Unavailable source will block the data warehouse view maintenance process. Evaluating queries at the data sources consumes local resources. Data warehouse view maintenance operations are not separated from OLTP operations.
Self-maintainable incremental maintenance	Data warehouse view maintenance operations are totally separated from OLTP operations. Unavailable source will not block the data warehouse view maintenance process. In the worst case, the number of rows accessed to maintain a view is the lowest.	Data are replicated at data warehouse. Need extra data storage for replicate data. Have to implement and maintain data transfer processes to transfer data from sources to data warehouse.
Not self-maintainable incremental maintenance	No replicate data at the data warehouse. No extra data storage for replicate data. Do not have to implement and maintain data transfer processes to transfer data from sources to data warehouse.	Unavailable source will block the data warehouse view maintenance process. Evaluating queries at the data sources consume local resources. Data warehouse view maintenance operations are not separated from OLTP operations. Have to design the view maintenance process carefully to avoid the anomaly problem. In the worst case the number of rows accessed is the highest. Performance is down-graded rapidly. Need extra storage for intermediate data (COLLECT tables).

space used in the data warehouse and the number of rows accessed in order to propagate an update to the target materialized view in the data warehouse. As the cost of data storage becomes remarkably low, this is the best approach to implement a data warehouse. However, in order to make the materialized views self-maintainable, the auxiliary views are stored in the data warehouse to provide the additional information. Extra storage and time overhead are therefore required to maintain the auxiliary views themselves. How to design materialized views at the data warehouse so that only necessary information are stored at the data warehouse is a major issue (Quass *et al*., 1996; Huyn, 1996a, b, 1997a, b).

A common variant of the view selection problem addressed in the literature earlier minimizes the sum of maintenance cost and query time on the view set. Converting what is inherently an optimization problem with multiple conflicting objectives into one with a single objective ignores the need and value of a variety of solutions offering various levels of trade-off between the objectives. We have scanned more than fifty research papers for this survey. Critical analysis is work out on the basis of type of algorithm or approach, type of constraints handelled or imposed. Critical analysis of some of the studies is given in the Table 2.

Table 2: Analysis of approaches and constraints

Approach	References	Constraints	Remarks
An evolutionary algorithm for materialized view selection based on multiple global processing plans for queries	(Zhang *et al*., 2001).	Data model is based on (SPJ) model rather than the multidimensional model. Proposed hybrid algorithms perform better than the heuristic algorithm in terms of cost savings, they often require longer computation time	Hybrid algorithms that combine the advantages of heuristic and evolutionary algorithms seem to perform the best
Proposed a heuristics algorithm based on individual optimum query plans	(Yang *et al*., 1997).	Without any resource constraint	Framework is based on specification of multiple view-processing plan (MVPP), which is used to present the problem formally
View maintenance techniques are classified into four major categories: self-maintainable recomputation, not self-maintainable recomputation, self-maintainable incremental maintenance and not self-maintainable incremental maintenance	(Wang *et al*., 2004).	Space usage and No. of rows accessed	Self-maintainable incremental maintenance performs the best in terms of both space usage and number of rows accessed.
A method for dealing with the problem was developed by formulating it as a state space optimization problem and then solving it using an exhaustive incremental algorithm as well as a heuristics algorithm.	(Theodoratos and Sellis, 1997).	That there are no space restrictions in the Data warehouse and space is not the problem does not discuss the complexity of the view selection problem	An exhaustive algorithm was designed and has provided heuristics for pruning the search space in different cases
Randomized approach based on the Simulated Annealing process	(Theodoratos *et al*., 2001).	Constraint that the new views and the old views together must be able to answer all the new queries has been imposed	Simulated Annealing has been used in a variety of optimization problems. In the Database area, it has been used for query optimization
The problem is formulated as a state space search problem by taking into account multiquery optimization over the maintenance queries and the use of auxiliary views for reducing the view maintenance cost.	(Theodoratos and Sellis, 1999).	Space constraints,	The static case of the DW design problem is studied
An exhaustive algorithm with greedy and heuristic algorithms that expand only a small fraction of the states produced by the exhaustive algorithm	(Ligoudistianos *et al*., 1998).	As the number of implications increases the r-greedy performs worse and the heuristic algorithm becomes the winner	r-greedy algorithms that prune the state space and a new heuristic algorithm that searches a small fraction of the state space and reports a sub-optimal solution
The problem noticed is NP-hard. A View Relevance Driven Selection (VRDS) algorithm based on view relevance	(Valluri *et al*., 2002).	The query processing cost and the view maintenance cost was taken in to consideration	VRDS algorithm performs better than greedy and MVPP algorithms when there is a space constraint and update frequency is high
By exploiting common sub	(Mistry *et al*., 2001).	Increase in cost of optimization	Extended the Volcano query optimization

Table 2: Continued

Approach	References	Constraints	Remarks
expressions between different view maintenance expressions is presented		is acceptable.	framework to generate optimal maintenance plans. A greedy heuristic has been proposed
The main contributions of this stufy are (a) an efficient view matching algorithm for views composed of selections, joins and a final group-by (SPJG views) and (b) a novel index structure (on view definitions, not view data) that quickly narrows the search to a small set of candidate views on which view-matching is applied.	(Goldstein and Larson, 2001).	1000 views in the system	View-matching algorithm was developed, including the filter tree, in SQL Server. An index structure is presented, called a filter tree
Local search technique. GLS approach to solve View Selection Problem has been adopted.	(Horng *et al*., 1999).	NP-complete problem	Genetic algorithm based solution
A new constrained evolutionary algorithm is proposed	(Yu *et al*., 2003).	Constraints are incorporated into the algorithm through a stochastic ranking procedure. No penalty functions are used.	Stochastic ranking, can deal with constraints effectively

IMPORTANT BOOKS, Ph.D THESIS AND RELEVANT LINKS

To help the other researchers in the materialized view domain, the collection of important books, Ph.D thesis and links related to the materialized view selection in data warehouse are given below:

Important books:

- Mastering Data Warehouse Design Relational and Dimensional Techniques By Claudia Imhopf
- The Data Warehouse Toolkit by Ralph Kimball
- Building a Better Data Warehouse by Don Meyer And Casey Cannon
- The Data Warehouse Life Cycle Toolkit by Ralph Kimball
- The Microsoft Data Warehouse Toolkit with SQL Server 2005 And Microsoft Business Intelligence Toolset by Ralph Kimball
- Building the Data Warehouse by W.H.Inmon.
- Effective Database Design Phi 1981

Important Ph.D Thesis:

- Materialized Views In Data Warehouses By Dallan Wendell Quass August 1997
- Selection and Maintenance of Views in a Data Warehouse by Himanshu Gupta September 1999
- Optimization Strategies for Data Warehouse Maintenance in Distributed Environments by Bin Liu April 2002
- Efficient Incremental View Maintenance for Data Warehousing By Songting Chen December 2005

Important links:

- **Designing a scalable data warehouse virtual lab:** www.msevents.microsoft.com
- http://www.icde2001.org
- www.area.cs.cnr.it
- www.codata.org
- www.infolab.stanford.edu/warehousing (whips prototype)
- www. rkimball.com (data warehouse training)
- www.dhise.andrews.edu/dw
- www.portal.acn.org/citation.cfm
- www.citeseer.ist.psu.edu
- www.peterindia.net/datawarehousinglinks.html
- **IBM research centre:** www.almaden.ibm.com
- www.sybase.com
- www.thedacs.com
- www.research.cornel
- www.web.mit.edu
- www.infolab.stanford.edu/warehousing

GLOSSARY OF TERMS AND BENCHMARK DATABASE FOR THE EVALUATION OF APPROACH

Here, key terms related to materialized views are discussed and the benchmark database links are given.

Glossary of terms:

- **Data warehousing:** A database system that collects and stores data from several databases is called as data warehouse
- **View:** A view is a derived relation defined in terms of base (stored) relation
- **Materialized view:** A view can be materialized by storing the tuples of the view in the database
- **View maintenance:** Just as a cache gets dirty when the data from which it is copied is updated; a materialized view gets dirty whenever the underlined base relations are modified. The potential cost incurred to propagate the changes in the view definition is the view maintenance cost
- **Incremental view maintenance:** Algorithms that compute changes to a view in response to changes to the base relations are called incremental view maintenance algorithms
- **Query processing:** It refers to the range of activities involved in extracting data from a database. It is concerned with choosing a strategy for processing a query that minimizes the amount of time necessary to compute the answer
- **Query optimization:** Query optimization refers to the process of selecting the most effective query evaluation plan for a query
- **Query evaluation:** Query is evaluated with the selected plan and the result of the query is the output
- **Cost of a query:** The cost of answering a query Q is the number of rows present in the table for that query Q_a, where Q_a is an ancestor of Q.
- **On-Line Transaction Processing (OLTP):** OLTP consists of parsing and translation, query optimization and query evaluation. It is concern with choosing a strategy for processing a query that minimizes the amount of time necessary to compute the answer
- **On-Line Analytical Processing (OLAP):** Queries involving group by and aggregation operators and provides excellent support for complex conditions, statistical functions and features for time series analysis
- **Query processing plan:** A sequence of primitive operations that can be used to evaluate a query is the query processing plan
- **Multiple views processing plan:** An alternative query processing plans used for evaluating a query
- **Base relation:** Base relation is a table consisting of columns representing the attributes and rows specifying the tuples or records
- **Query frequency:** It is the frequency of posing a particular query on a base relation
- **Updation frequency:** The frequency at which the base relation is updated (insert, delete and modify)

Benchmark database for the evaluation of approaches:

- A free, open source, performance benchmark for private testing of relational database systems. Tests are written in the C language using GNU tools:

  www.osdb.sourceforge.net

- The TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications:

  www.it.iitb.ac.in/~chetanv/personal/acads/db/report_ht ml/node3.html

- The TPC-D benchmark, which simulates a complex DSS workload with 17 queries areas including data warehousing, high performance OLTP and Web/E-Commerce:

  www.taborcommunications.com/dsstar/98/1110/1004 06.html

- The TPC-E benchmark simulates the data processing associated with a real warehouse:

  www.itjungle.com/two/two080807-story04.html

- http://www.tpc.org

POSSIBLE APPROACHES

To retrieve the information relevant to the query form the data warehouse, much more time is required because the data warehouse is the huge storage of organized data. For the retrieval of relevant information, particular data have to be located in the data warehouse. To locate this, whole data warehouse may have to be explored i.e. query related base relations have to be located and the relevant data may get displayed. For each query this task will be repeated. As the data warehouse contents huge data and the number of queries may be any value, for the execution of all the queries much higher time is required. To avoid this, if we find one particular requirement of all the queries and then keeping only the particular base relations or copy of these base relations, where from we can get the relevant information. To do this, we may have to search all

the queries. In this regard one can use the different search strategies.

In the case of base relation updation, materialized view selection may get affected. So, it does not mean that once find solution will be the last and final solution, we have to take care of the base relation updation. Now the materialized view selection becomes the load balancing problem. How it can be the load balancing problem? In this case, we have to balance the base relation updation cost with query processing cost to achieve the optimum materialized view. Materialized view selection problem can be formulated as the load balancing problem and can be solved by the dynamic programming technique. Other version of this problem is that it is the optimization problem. In this case, one can consider this problem as multi objective (two) problem i.e., minimization of base relation updation cost and other objective will be minimization of the query processing cost.

Numbers of advanced techniques are available to solve the searching and the optimization problem. Here in this problem, both the scopes are available. Different possible solutions are discussed below which may give the solution to materialized view selection problem. This problem can be formulated as the neural network problem. Different queries can be used as the input and the relevant base relation index with the particular view will be the output. And later this neural network can be tested for any random query. If we consider this problem as the optimization problem then the ant colony optimization technique can also be applied to find the shortest path. Here the shortest path is the collection of minimum cost view of particular query. This can be done locally or globally. In addition to this swarm optimization can be used. It can also be solved by genetic algorithm by treating the materialized view selection problem as multi objective problem. In case of query searching, different searching techniques can be used e.g., spiral search, Tabu search, circulant matrices etc.

CONCLUSION

We have analyzed more than fifty research papers and books. Through this study we tried to give the basic content which is required to be known to the beginner who is going to work in this domain. We have discussed about the basic mathematical background including the cost model formulation. We have critically analyzed the past and present methods or techniques to solve the materialized view selection problem by providing the summary in Table 1 and 2. We have also provided the details of books, thesis, important links, glossary of terms and the benchmark database which can be used to check anybody's approach or technique. In addition to these we have proposed some solutions based on the new advanced techniques of searching and optimization such as Spiral search, Tabu search, Circulant matrices, ant colony optimization, swarm optimization, genetic optimization. Dynamic programming can also be the base to solve this problem as a load balancing problem.

One typical area of this problem is the cost model formulation i.e., mathematical formulation. In literature till date every one used the linear cost model. We think that here also some scope available to formulate the problem as the piecewise linear cost model. In place of having only two objectives i.e. view maintenance cost and query processing cost, we can have some more depth of these objective function by representing them as the piecewise linear functions. It is beyond the scope of this study to discuss all possible approaches in detail. Few of the approaches are brief out in the previous section and the present section. Finally, we can say that this study can be the initial guide for the beginner in this domain. Recently many researchers are working in this domain. Still there is a scope to contribute much more in this domain.

REFERENCES

Baralis, E., S. Paraboschi and E. Teniente, 1997. Materialized view selection in a multidimensional database. Very Large Data Bases Proceedings of the 24th International Conference on Very Large Data Bases. 1997, Morgan Kaufmann Publishers Inc., San Francisco, pp: 488-499.

Blaiseley, J.A., P. Larsonn and E.W. Tompa, 1986. Efficiently updating materialized views. ACM SIGMOD Record, 15: 61-71.

Chaudhuri, S. and U. Dayal, 1997. An overview of Data Warehousing and OLAP technology. ACM SIGMOD Record., 26: 65-74.

Cui, Y. and J. Widom, 1999. Storing Auxiliary Data for Efficient View Maintenance and Lineage Tracing. http://www-db.stanford.edu/pub/papers/auxview.ps Technical Report, Stanford University, 1999.

Davis, L. and D. Smith, 1987. Genetic Algorithms and Simulated Annealing: An Overview. In: Genetic Algorithms and Simulated Annealing, Davis, L. (Ed.). Pitman, London, pp: 1-11.

Dhote, C.A. and M.S. Ali, 2007. Materialized view selection in data warehousing. Fourth International Conference on Information Technology, ITNG 2007, April 2-4, IEEE Computer Society Washington, DC. USA., pp: 843-847.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. 1st Edn. Addison- Wesley Longman Publishing Co. Inc., MA, USA., ISBN: 0201157675 pp: 372.

Goldstein, J. and P.A. Larson, 2001. Optimizing queries using materialized views: A Practical, scalable solution. ACM SIGMOD, 30: 331-342.

Griffin, T. and L. Libkin, 1995. Incremental maintenance of views with duplicates. ACM SIGMOD Record, 24: 328- 339.

Gupta, H. and I.S. Mumick, 1996. Selection of Views to Materialize Under a Maintenance Cost Constraint. 1st Edn., Springer, Berlin/Heidelberg, ISBN: 978-3-540- 5452- 0.

Gupta, H., 1997. Selections of Views to Materialize in a Data Warehouse. 1st Edn., Publisher Springer, Berlin/Heidelberg, ISBN: 978-3-540-62222-2.

Gupta, H., V. Harinarayan, A. Rajaraman and J.D. Ullman, 1997. Index selection for OLAP. Proceeding of International Conference on Data Engineering (ICDE' 97), April 7-11, Birmingham, UK., pp: 208-219.

Gupta, H., 1999. Selection of Views to Materialize Under a Maintenance Cost Constraint. 1st Edn., Publisher Springer, Berlin/Heidelberg.

Harinarayan, V., A. Rajaraman and J. Ullman, 1996. Implementing data cubes efficiently. ACM SIGMOD Record, 25: 205-216.

Horng, J.T., Y.J. Chang, B.J. Lin and C.Y. Kao, 1999. Materialized view selection using genetic algorithms in a data warehouse system. Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, Dept. of Computer Science and Inf. Eng., Nat. Central University, Jungli, pp: 22-27.

Hull, R. and G. Zhou, 1996. A Framework for supporting data integration using the Materialized and virtual approaches. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996, ACM New York, USA., pp: 481-492.

Huyn, N., 1996a. Efficient self-maintenance of materialized views. http://dbpubs.stanford.edu/pub/1996-3, 1996.

Huyn, N., 1996b. Efficient view self-maintenance. In Proceedings, Workshop on Materialized Views: Techniques and Applications, http://140.115.82.191/old/warehouse/cqvsm.ps.

Huyn, N., 1997a. Exploiting dependencies to enhance view self-maintainability. Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, Publisher Morgan Kaufmann Publishers Inc., San Francisco, CA, USA., pp: 26-35.

Huyn, N., 1997b. Multiple-View self-maintenance in data warehousing environments. Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, Publisher Morgan Kaufmann Publishers Inc., San Francisco, CA, USA., pp: 26-35.

Inmon, W.H., 1996. Building the Data Warehouse. 2nd Edn., John Wiley and Sons, Canada, ISBN: 0471-14161-5.

Ishibuchi, H., T. Murata and S. Tomioka, 1997. Effectiveness of genetic local search algorithms. Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, Publisher Morgan Kaufmann, USA, July 19-23.

Kolen and E. Pesch, 1994. Genetic local search in combinatorial optimization. Discrete Applied Math. Combinatorial Operat. Res. Comput. Sci., 48: 273-284.

Korth, H.F. and A. Silberschatz, 1986. Database System Concepts. 4th Edn., McGraw-Hill, New York, ISBN: 0-07- 120413-X.

Labio, W., D. Quass and B. Adelberg, 1997. Physical database design for data warehouses. 13th International Conference on Data Engineering Proceedings, April 7-11, Birmingham, UK., pp: 277-288.

Lee, M. and J. Hammer, 2001. Speeding up warehouse physical design using a randomized algorithm. Int. J. Cooperative Inform. Syst., 10: 327-353.

Liang, W., H. Wang and M.E. Orlowska, 2001. Materialized view selection under the maintenance time constraints. Data Knowledge Eng., 37: 203-221.

Ligoudistianos, S., D. Theodoratos and T. Sellis, 1998. Experimental evaluation of data warehouse configuration algorithms. Proceedings of the 9th International Workshop on Database and Expert Systems Applications, August 1998, Department of Electric and Computer Eng., Nat. Tech. University of Athens, pp: 218-223.

Merz, P. and B. Freisleben, 1997. A Genetic local search approach to the quadratic assignment problem. The Seventh International Conference on Genetic Algorithms, July 19-23, Michigan State University, East Lansing, Michigan, pp: 1-1.

Mistry, H., P. Roy, S. Sudarshan and K. Ramamritham, 2001. Materialized view selection and maintenance using multi query optimization. ACM SIGMOD Record, 30: 307- 318.

Mohania, M., 1997. Avoiding re-computation: View adaptation in data warehouses. Proceedings of 8th International Database Workshop, 1997, Hong Kong, pp: 151-165.

Mohania, M., S. Konomiy, Y. Kambayashiz and M. Vincentx, 1999. Designing view maintenance algorithm in data warehousing environment. Proceedings of the 9th International Conference on Management of Data (COMAD), pp: 117-133.

Quass, D., 1996. Maintenance expressions for views with aggregation. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, June 4-6, ACM Press, Quebec, Canada, pp: 110- 118.

Quass, D., A. Gupta, I.S. Mumick and J. Widom, 1996. Making views self-maintainable for data warehousing. 4th International Conference on Parallel and Distributed Information Systems, December 18-20, Department of Computer Science Stanford University, CA., pp: 158-169.

Roussopoulos, N., 1982. View indexing in relational databases. ACM Trans. Database Syst., 7: 258-290.

Sellis, T.K., 1988. Multiple query optimization. ACM Trans. Database Syst., 13: 23-52.

Shim, K.,T. K. Sellis and D. Nau, 1994. Improvements on a heuristic algorithm for multiple-query optimization. Data Knowledge Eng., 12: 197-222.

Shukla, A., P.M. Deshpande and J.F. Naughton, 1998. Materialized view selection for multidimensional datasets. Proceedings of the 24th International Conference on Very Large Data Bases, 1998, Morgan Kaufmann Publishers Inc., San Francisco, CA. USA., pp: 488-499.

Smith, J.R., C. Li, V. Castelli and A. Jhingran, 1998. Dynamic assembly of views in data cubes. Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1998, Seattle, Washington, United States, pp: 274-283.

Theodoratos, D. and M. Bouzeghoub, 1999. Data currency quality factors in data warehouse design. Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99), June 14-15, Heidelberg, Germany, pp: 1-1.

Theodoratos, D. and T. Sellis, 1999. Designing data warehouses. Data Knowledge Eng., 31: 279-301.

Theodoratos, D. and T. Sellis, 1997. Data warehouse configuration. Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, Morgan Kaufmann Publishers Inc., San Francisco, CA. USA., pp: 126-135.

Theodoratos, D., S. Ligoudistianos and T. Sellis, 1999. Designing the global data warehouse with SPJ views. Proceedings of the 11th International Conference on Advanced Information Systems Engineering, June 1999, Springer-Verlag Lecture Notes in Computer Science, pp: 180-194.

Theodoratos, D., T. Dalamagas, A. Simitsis and M. Stavropoulos, 2001. A randomized approach for the incremental design of an evolving data warehouse. Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling, November 2001, Springer-Verlag, Lecture Notes in Computer Science, pp: 325-338.

Valluri, S.R., S. Vadapalli and K. Karlapalem, 2002. View relevance Driven Materialized View Selection In Data Warehousing Environment, Australian Computer Science Communications. 1st Edn., IEEE Comput. Soc., Los Alamitos, CA, USA.

Wang, X., L. Gruenwalda and G. Zhu, 2004. A performance analysis of view maintenance techniques for data warehouses. Data Warehouse Knowledge, 2004.

Yang, J., K. Karlapalem and Q. Li, 1997. Algorithms for Materialized view design in data warehousing environment. Proceedings of the 23rd International Conference on Very Large Data Bases, 1997, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA., pp: 136-145.

Yu, J.X., X. Yao, C.H. Choi and G. Gou, 2003. Materialized view selection as constrained evolutionary optimization. IEEE Trans. Syst. Man Cybernetics Part C, 33: 458-467.

Zhang, C., X. Yao and J. Yang, 1999. Evolving materialized views in data warehouse. Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99, School of Computer Science, New South Wales University, Canberra, ACT, pp: 829-829.

Zhang, C., X. Yao and J. Yang, 2001. An Evolutionary approach to materialized views selection in a data warehouse environment. IEEE Trans. Syst. Man, Cybernetics Part C, 31: 282-294.