EFFICIENCY, PERFORMANCE AND VIEW MATERIALIZATION IN OBJECT-ORIENTED PARADIGM

Noreen Ashraf[#,*,1], Salman Afsar[#,*,2], Muhammad Asam[#,*,3]

[#,*,1] Department of Computer Science, National College of Business Administration And Economics, Lahore, Pakistan

[#,*,2] Department of Computer Science, University of Agriculture, Faisalabad, Pakistan

[#,*,3] Department of Computer Science, GC University, Lahore, Pakistan

Corresponding author: noreen.ashraf48@yahoo.com

ABSTRACT: *Certain aspects of the incremental maintenance of MVs have been studied in detail for relational database management systems (DBMSs) as well as in a deductive setting . Work on an object-based setting is significantly hard to find. The solution to the incremental view maintenance problem presented here assumes as do all other solutions the availability of the update event, the changes made to the database and the current materialized state of the view. . The VMOP (View Materialization in Object-Oriented Paradigm) is an implementation to the solution to the IVM problem. The solution yields an Incremental Maintenance Plan (IMP) algorithm for object based solution . The result of an experimental performance evaluation of the VMOP provides solution to the incremental maintenance OQL views. So that it is easier to integrate the solution into the kind of query processing frameworks that normal database management system depend on."*

Abstract- It's a challenging task to develop socially interactive agents .

Keywords: Materialized View, Incremental Maintenance Plan, VMOP, OQL .

1. INTRODUCTION

Extensive research has been dedicated to database management systems during the last three decades. The main objective of that research was to develop abstract and logical modals for specifying the structure of data stored in database.[1] Views form the external level representing the interest of database user groups. A view provides the means for logical data independence. A view is a definition of a derived relation the extent of which is determined by a query expression. Every time a view is used in query its extents is re-computed. The term data independence appears in ANSI/SPARC three schema architectures. Where views are external level represents the interest of different user groups [2]. A view is a definition of a derived relation the extent of which is determined by a query expression. Every time a view is used in query its extents is re-computed. Sometimes such views are referred as virtual. This research then provides the bases for efficient techniques for query optimization, indexing, searching, storage and retrieval. Query performance can be still improved in DMBS [3].

The information requested by the user is assembled on-demand. Research has been made to develop methods for efficiently assemble such information whenever required (Kuno 1996)[4]. After assembling the information it is then provided to the user. If the information required is defined as a view than a query over a view requires assembling the extent of the view. That extent of the view is materialized and than used in query answering. In that way it saves the cost of assembling it each time. When the extent of the view is materialized it is referred as materialized view. In MV there is initial cost in assembling the information after that it is just refreshed the MV whenever updates change the relations from which the view is derived.

MV is an important part of DBMS as improving the performance of query processing.MV has gained a lot of interest in the database community for their application in online analytical processing (OLAP), data warehousing, data integration and replication. A data ware house can be considers as a collection of MVs over the data stored in information sources and the problem of maintaining a data warehouse can b seen as the problem of maintaining such views.[1]. Actually through the system aspects of the data base we can view or analyze the performance of incremental view maintenance ,the size of the database updates or refurnished and the size of the relation involve. Database query optimizer is an appropriate components of the database system to decide whether a view in maintained incrementally or not. Because all the parameters that may affect that choice are already in the knowledge of query optimizer. Incremental views are little supported by experimentation to prove that they are beneficial than non-incremental views. [5].

The usage of materialized objects oriented views in object relational database warehousing system is most reliable .A novel technique named *hierarchical materialization* is proposed by us for the materialization of method results in object oriented views .This technique was implanted and evaluated by a large number of people who were concerned with the methods without input arguments and with input arguments as well .The results showed the *hierarchical materialization* reduces method re-computation time .Materialization method with input arguments introduces only a small time overhead [6]. To speed up queries ,the Materialized views have been found very effective and these are also supported a lot by commercial database and data warehouse system. All these methods or ways need wel organized methods to maintain Materialized views [7].

For data integration, application and high performance query processing, view materialization is an efficient technique. Increment maintaining materialized views have very relevant solution for all the problems. So far, most work on this problem has been confined to relational settings and solutions have not been comprehensively evaluated [1]. Materlized views present a complex and thoughtful process. to select the optimum set of materialized views a new algorithms is also proposed. This new proposed algorithm sets this set based utilization rate, relationship update rate cost calculation.[8]

We come to across many major differences between object-oriented paradigms and relational that can be seen when addressing the object-oriented view materialization problem .

- Most of the work done on IVM is in a relational setting.

- A few proposals have addressed IVM in the context of object oriented database systems.
- The study of MV shows that they are applicable in many applications.

2. LITERATURE REVIEW

View materialization is known to be a valuable technique for performance optimization in relational databases, and much work has been done addressing the problem of consistently maintaining relational views under update operations. However, little progress has been made thus far regarding the topic of view materialization in object-oriented databases (OODBs). They demonstrated that there are several significant differences between the relational and object-oriented paradigms that can be exploited when addressing the object-oriented view materialization problem. They used the subsumption relationships between classes to identify branches of classes to which we do not need to propagate updates. Similarly, they used encapsulated interfaces combined with the fact that any unique database property is inherited from a single location to provide a registration/notification service for optimizing incremental view updates .

They have successfully implemented all proposed techniques in the MultiView system, which provides updatable materialized classes and virtual schemata on top of the GemStone OODBMS. They also reported results from the experimental studies have run on the MultiView system measuring the impact of various optimization strategies incorporated into our materialization update algorithms (Rundensteiner 1996)[4] .

Selecting views to materialize is one of the most important decisions in designing a data warehouse. This paper present a framework for analyzing the issues in selecting views to materialize so as to achieve the best combination of good query performance and low view maintenance. They first develop a heuristic algorithm which can provide a feasible solution based on individual optimal query plans . The materialized view designs problem as 0-1 integer programming problem, whose solution can guarantee an optimal solution .

They stated that there are two approaches towards providing integrated access to multiple, distributed, heterogeneous databases: (1) lazy or on-demand approach (2) data warehousing approach. The specific contributions of their paper are as: A framework is presented to highlight issues of materialized view design in a distributed data warehouse environment. This framework is based on the specification of Multiple View Processing Plan (MVPP) which is used to present the problem formally. They provide two algorithms to generate MVPP(s): one can generate a feasible solution expeditiously; the other can provide an optimal solution by mapping the optimal MVPP generation problem as an O-l integer programming problem. They have addressed and designed algorithms for the materialized view design problem. The work presented is the outcome of the first stage of research in Materialize View Design project. Their focus on developing an analytical model for a multiple view processing environment to simulate different scenarios to evaluates the solutions for the materialized view design problem [9] .

Addressing the object-oriented view materialization problem. They demonstrate that there are several significant differences between the relational and object-oriented paradigms that can be exploited when addressing the object-oriented view materialization problem. First, propose techniques that prune update propagation by exploiting knowledge of the subsumption relationships between classes. Second, use encapsulated interfaces , Third, introduce the notion of hierarchical registrations.

They have successfully implemented all proposed techniques in the MultiView system on top of the GemStone OODBMS. Their paper also present a cost model for our update algorithms and report results from the experimental studies run on the MultiView system, measuring the impact of various optimization strategies incorporated into our materialization update algorithms (Rundensteiner 1998)[10].

An application is built, an underlying data model is chosen to make that application effective. The naïve solution of copying the underlying data and modeling is costly in terms of storage and makes data maintenance and evolution impossible. The technique enables applications to customize shared data objects without affecting other applications that use the same objects. they reduce the need to re-compute the view and/or data being queried,they speed up the querying of large amounts of data. Further, they provide a systematic way to describe how to re-compute the data, maintenance and evolution can be automated. Materialized views are especially useful in data warehousing, query optimization, integrity constraint maintenance, online analytical processing, and applications such as billing, banking, and retailing (Gupta 1999)[11] .

The development of techniques for supporting incremental maintenance of materialized views has been an active research area for over twenty years. They present the results of an experimental performance analysis carried out in a system that incrementally maintains OQL views in an ODMG compliant object database. The results indicate how the effectiveness of incremental maintenance is affected by issues such as database size, and the complexity and selectivity of views. MV is an important part of DBMS as improving the performance of query processing. So far, most work on this problem has been confined to relational settings and solutions have not been comprehensively evaluated. (Ali 2000)[1]

Materialized views are supporting to make view faster and efficient in terms of quires and are being used by data warehouses and other database systems. A framework was developed to integrate multiple choices in an organized and effective way. To maintain the workload of different queries and updates these algorithms may also be used in materialized views.

For the maintenance of set of materialized views they found an efficient plan by utilizing common sub-expressions among many view maintenance expressions. A framework was developed that was utilized to integrate the multiples choices in an effective and organized way. They evaluated by using different techniques that many-fold improvement can be made in view maintenance. To speed up workloads

containing queries and updates these algorithms may also be utilized in materialized views .

They worked to find out the ways to reduce the cost of view maintenance plans by utilizing transient materialization of ordinary sub-expressions. To speed up maintenance of the given views by additional expressions may be chosen. The ways to determine the optimal maintenance plan for each view were found.

A greedy heuristic was proposed which iteratively picks up views to decrease the overall materialization cost by picking up iteratively in order. At present, there are less options available for maintenance point of view in Data warehouses and data marts as they maintain automatically. The aim is to find best way to maintain the materialized views which has its own importance. By applying different techniques on existing optimizer, a performance study was conducted to study their benefits. In another direction, to speed up a workload of queries by selecting materialized view.

A modification can be applied to the greedy algorithm as follows: candidates would be intermediate/final results of queries, and advantages to queries would be included when computing advantages(Mistry 2001)[7]. Described that multi-query optimization using heuristic is practical, and provides major advantages. Three cost-based heuristic algorithms were proposed: Volcano-RU and Volcano-SH, and a greedy heuristic. Newly designed algorithms in this research work can easily added to existing optimizers. A performance study is presented by comparing the algorithms, using workloads consisting of queries from the TPC-D benchmark. Our implementation describes that the algorithms can be added to an existing optimizers with a reasonably less effort. This performance study, using queries based on the TPC-D benchmark, express that multi-query optimization is practical and gives significant benefits at a reasonable cost.Multi-query optimization was also demonstrated on a real database system to check the advantages. As a result, it is aimed that the groundwork have been laid for convenient use of multi-query optimization, and multi-query optimization will form a criticalpart of all query optimizers in the future. (Roy 2000)[12]

In object-relational data warehousing systems is promising in the application of materialized object-oriented views. In object-oriented views, a technique was purposed for materialization of method called hierarchical materialization.When an object used to materialize the result of method m is updated, and then m has to be recomputed. This recomputation can utilized unaffected intermediate materialized results of methods called from m, thus reducing a recomputation time. It revealed in results that hierarchical materialization reduces method recomputation time. Moreover, materializing methods with input arguments of narrow discrete domains introduces only a small time overhead (Bebel 2001)[6].

Scalable and fastalgorithmis establish whether part or all of a query can be computed from materialized views and demonstrated how it can be incorporated in transformation-based optimizers.They show an efficient view-matching algorithm for SPJG views and described its integrated into a transformation-based optimizer . They also show an index structure, called a filter tree, which efficiently speeds up the search for applicable views. On implementation in Microsoft SQL Server, it found that obtained experimental results of the algorithm are fast and scales to very large numbers of views. In future work, plan is to extend the algorithm to cover a broader class of views and substitute expressions **(**Goldstein and Larson 2001)[13].

In the concept of materialized view is quite common and significant in the environment of data warehouse environment these days. It has clearly an objective efficiently maintaining and supporting the processing of OLAP query system. Most of the time, these materialized views are derived from the select-project-join of a number of base relations. An efficient incremental view maintenance strategy is given the name of "delta propagation." This strategy can minimize the total size of base relations by analyzing the properties of the relations. The strategy contains the delta expression and delta propagation tree which needs to be defined. The algorithm which can find the optimal delta expression is proposed since this algorithm is dynamic programming carrier. Several experimental results show the usefulness and efficiency of this strategy (Lee 2001)[14].

The well known concept which has been quite sufficiently addressed and established in literature and implemented in database products as well is of "incremental view maintenance". This view is further implied upon all aggregate functions in a form of well established mechanism. This mechanism excludes those aspects and features which are not distributive over all operations. The optimization of this view is possible in two different ways.

The first way can be only the re-computing of the affected groups and secondly, by extending the infrastructure of incremental work. It is done to maintain and support those functions that are algebraic in nature. The further optimization computing is performed when multiple but dissimilar in nature aggregate functions are computed under the same view. The other important conflicting issues of incremental views of maintenance are addressed which are related to super aggregates. These include as well the materialized OLAP cubes. The implementation of our algorithm on the prototype of IBM DB2 UDB has proved the validity of our approach with the help of an experimental evaluation (Palpanas 2002)[15].

The effective and high processing of of query, data integration and replication is possible with the technique of view materialization. In the ODMG- compliant data bases, the incremental maintenance of materialized OQL views are solved with the help of MOVIE which is a complete, evaluated and implemented solution of all the problems that come in the way. In object data base, the IVM problems are the best sort out. The main contribution of the paper can be best synthesized as:

(1) Any update operation in ODMG language binding can be best handled with the ODMG-compliant schemas since these are the solutions to the problems of incremental maintenance of materialized OQL views.

(2) The yielding of performance benefits under certain circumstances can be understood by the experimental evaluation of the implemented system (Ali 2003)[16]**.**

To compute the answers to the queries like the size of the view-set on a given database, the solution must be found in

advance. For conjunctive queries and their workload, the decidability and complexity of the problem must be explored. If the workload queries have self-joins, the result differ significantly. The dynamic programming algorithm is provided which finds the minimal size of disjunctive view-sets without any self-joins. The discussion is also done on heuristics about the efficiency of algorithm. Thus, efforts are done for finding out the optimal solution (Chirkova 2003)[17].

Materialized views in the context of multidimensional databases (MDDBs). A materialized view is a view whose content is explicitly stored in the database. They proposed efficient incremental maintenance techniques. MDDBs are an ideal environment for materialized views because frequency of updates is low, MDDB data models permit easy adoption of incremental maintenance, and queries can be modeled in such a way to allow an easy definition of the view selection problem, i.e., the problem of selecting which query to materialize in an MDDB. Hence present the problems of choosing and maintaining materialized views with the corresponding solutions (Paraboschi 2003)[18].

An efficient incremental maintenance for multiple joint views. The delta propagation strategy to multiple views has been extended. The shared common intermediate results among views can be shown effectively by the recursive property of this strategy . The whole process can be done by first defining the multiple view maintenance problems and then applying the heuristic algorithm which can find global maintenance plan for the under consideration views (Lee 2005)[19].

The views stored in data base ware house must be kept updated and current. It is necessary . They have the ability to develop change table technique in order to maintain incremental views expressions by involving rational and aggregate operators. The change table technique performs on previously proposed techniques by orders of magnitude. The maintaining views expressions are effectively and efficiently extended by developing framework for outer-join operators. The developed change table technique has clearly shown and proved that this is an optimal incremental maintenance scheme for the given view expression under the heading of reasonable assumptions Gupta 2006)[20].

The efficient selection of materialized views can be achieved simply by three basic factors: By estimating the query cost, by view maintenance cost, or by application of heuristics. This approach is particularly helpful in some cases . In order to minimize query response time, it has been proved that by deciding which set of views in the data cube must be materialized. The alternative ways of evaluating multiple queries and views, sub expression and sharing can be sort out by utilizing AND/OR DAG expressions. For the better performance of the data warehouse, proposed approach can be applied to optimize the views Dhote 2007)[21].

The issue that a data warehouse mostly integrates the businesses information and data from inner and outer data sources . For this issue, a new algorithm has been proposed in this paper for the efficient selection of optimum set of materialized views. This algorithm has been based upon certain factors namely, utilization rate, relationship update rate and calculation cost. For the real life application of this algorithm, the development of this work was produced within the domain of medical project. It was done in order to find validity of this newly proposed algorithm (Encinas 2007)[8].

The view that to speed up entire data ware housing process which is constrained by storage and issues of cost consideration, this newly proposed algorithm can help in selecting efficient and proper set of materialized views. For the efficient gain and loss metrics, a cost model for data warehouse query and maintenance along with efficient view selection algorithms has been derived. The most important aspect of this paper is the process of speeding up the materialized views. This will in return, reduce the overall cost of data ware house and maintenance issues (Hung 2007)[22].

If the strategy of materializing views is based upon cache, it further reduces the cost of views refreshment on the basis of greedy and dynamic selection algorithms. For the suitability of variety of queries, the application of views refreshment is more appropriate in contrast to greedy algorithm. The efficiency of views in materialized set can be low if there is frequent substitution. This can be avoided by preferring the cache- updating system over dynamic selection algorithm (Yin 2007)[23].

Materialized views can be speed up by processing of the query greatly. But it must be considered that they have to be kept up to date and useful. They have represented a very innovative and novel way to maintain the materialized views which are responsible of relieving the updates of this overhead. It is proposed by this approach that maintenance can be postponed until the system has free cycles or the view has been reference by a query .

(1) While ensuring that that the queries will only see up to date views, they introduced a new approach for maintaining materialized views that relieves the updates of view maintenance.

(2) To obtain simple and efficient maintenance expressions, they have exploited new versions.

(3) By merging multiple maintaining tasks for a view and by eliminating redundant updates of the same row, they reduced the cost of view maintenance.

(4) They have used low priority background jobs for exploiting the system free cycles to maintain views. The view is turned immediately up to date when the query demands it.

(5) For the demonstration of the feasibility and benefits of this approach, a prototype implementation in SQL Server 2005 has been proposed with extensive experiment **(**Zhou 2007)[24]**.**

In large database specifically in distributed database, query response time plays an valuable role as timely access to information and it is the basic requirements of successful business application. A data warehouse uses multiple materialized views to efficiently process a given set of queries . A speedy response time and appropriateness are important factors in the success of any database. It is impossible the materialization of all views because of the space constraint and maintenance cost constraint. Choosing of materialized view is one of the most important decisions in designing in data warehouse for optimal efficiency. Choosing a suitable set of views that minimizes the total cost associated with the materialized views and is processing . This paper

gives the results of proposed tree based materialized view selection algorithm for query processing .In distributed environment where database ID distributed over the nodes on which query should get executed and also plays an important role .this paper also proposes mode selection algorithms for fast materialized view selection in distributed environment .And finally it is found that the proposed methodology performs better for query processing as compared to other materialized view selection strategies (Mr. P. P. Karde 2010)[25].

A speedy response time and appropriateness are important elements in the success of any database .in large database specifically in distributed database . Query response plays an valuable role as timely access to information and it is the basic needs of successful business application . A data warehouse uses multiple materialized views to efficiently process a given set of queries .it is impossible ,the materialization of all views because of the space constraint and maintained cost constraint . materialized view selection is one of the critical decisions in designing a data warehouse for optimal efficiency. Choosing a accurate set of views that minimizes the total cost associated with the materialized views is the key parts I data warehousing. Materialized views are found useful for query processing. This paper gives an overview of various techniques that are implemented in past recent for selection of materialized view. The issues related to maintaining the materialized view are also discussed in this paper (Mr. P. P. Karde 2010)[26].

A lot of different views can be made and materialized from data warehouses per the user requirements specified in the queries being generated against the information contained in the warehouses. Choosing views to be materialized is one of the most important decisions in designing a warehouse. Because the two change in user needs and constraints aver time. View definitions stored in a data warehouse are dynamic in nature.

This paper specified on the issue of materialized views in data warehousing to enable efficient information management. This carries selection maintained and updating of materialized views and how these issues create on impact in business scenarios (G. Prabagaran 2013)[27].

In order to fulfill the user's requirement in the dynamically changing data warehouse environment materialized view evolve. So that, evaluation approach of materialized view focuses on selecting materialized views in the design process of data ware houses or in response to data changes or to data sources changes and sometimes to keep a check on the DW quality under schema evolution .although this materialized view evolution problem for evolving an appropriate set of views is addressed by few researches. In order to identify their advantages and disadvantages, none of the surveys provides a classification of materialized view evolution approaches. This survey tries to fill this gap .the present paper provides a review of model based materialized view evolution methods by identifying the three main dimensions namely (i) Model/Design Model (ii) Architecture (iii) Framework (Anjana Gosain 2015)[28].

3. MATERIALS AND METHODS

3.1. The OO7 Database Schema

The OO7 benchmark schema is designed to be indicative of many complex application domain like computed aided design, computer aided manufacturing and computer aided software engineering(Ali 2000)[1] . It is designed for evaluating different aspects of database system performance and has been widely used in performance analysis of ODBMs. That is why we have chosen the OO7 database schema to be used throughout for illustrating different features of ODMG standard, the lambda-Db system and for performance evaluation of the VMOP system (Colby 1996)[29]. Among the performance characteristics tested by OO7 are:

- The speed of many different kinds of pointer traversal including traversal over cached data, traversals over disk resident date, spare updates, updates of cached data and the creation and deletion of objects.
- The performance of the query processor on several different types of queries.

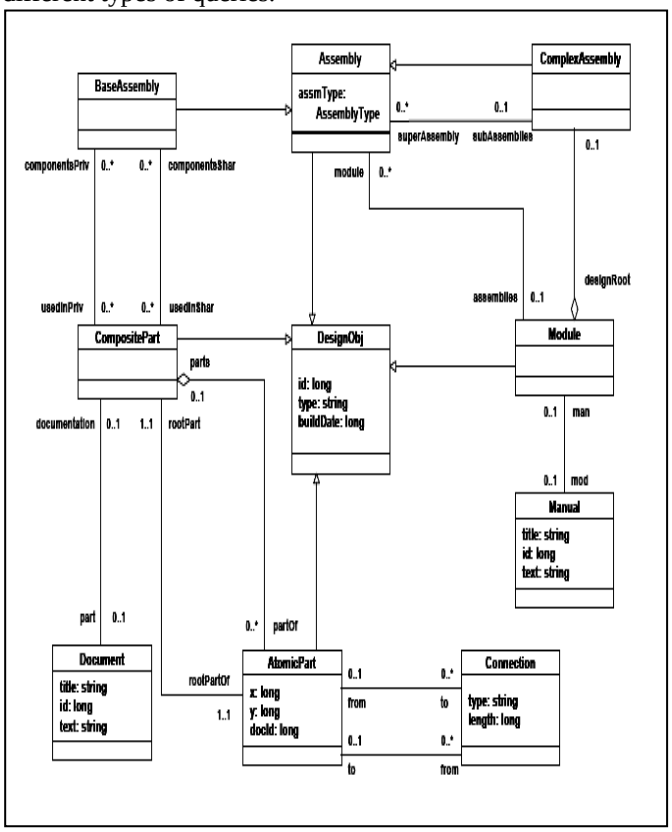

Fig 1.OO7 Modal in UML

There are three sizes of the OO7 benchmark, small, medium and large.

Table 2. Parameters of OO7

Parameter	Small	Medium	Large
Number of atomicparts per compositePart	20	200	200
Numberof connection per AtomicPart	3,6,9	3,6,9	3,6,9
Document Size	2000	20000	20000
Manual size	100KB	MB	1MB
Number of composite parts per Modules	500	500	500

Number of Assemblies per complesAssembly	3	3	3
Number of Assembly at each level	7	7	7
Number of CompositeParts per Assembly	3	3	3
Number of Modules	1	1	10

4. 3.2. THE OBJECT-ORIENTED DATABASE (OODB) MODEL

Object-oriented database systems, which can be considered fifth-generation database technology, began developing in the mid-80's out of a necessity to meet the requirements of applications beyond the data processing applications, which characterized relational database systems (fourth-generation database technology) . Attempts to use relational database technology for advanced applications like computer aided design (CAD), computer aided manufacturing (CAM), software engineering, knowledge-based systems, and multimedia systems, quickly exposed the shortcomings of relational database systems. The need to perform complex manipulations on existing databases and a new generation of database applications generated a need that would be better satisfied by object-oriented databases (OODBs)(Chirkova 2003)[17] .

Many definitions of object orientation and object-oriented databases have been developed over the years but object oriented databases as databases that integrate object orientation with database capabilities. Object orientation allows a more direct representation and modeling of real world problems, and database functionality is needed to ensure persistence and concurrent sharing of information in applications. Most current OODBs are still not full-fledged database systems comparable to current relational database systems RDBs Object-oriented database systems evolved from a need to satisfy the demand for a more appropriate representation and modeling of real world entities, so OODBs provide a much richer data model than relational databases (Lee 2001)[14]. The OODB paradigm is based on a number of basic concepts, namely object, identity, class, inheritance, overriding, and late binding. In the object-oriented data model (OODM), any real world entity is represented by only one modeling concept – the object . An object has a state and a behavior associated with it. The state of an object is defined by the value of its properties attributes (Chirkova 2003)[17]. Properties can have primitive values (like strings and integers) and nonprimitive objects. A nonprimitive object would in turn consist of a set of properties. Therefore objects can be recursively defined in terms of other objects. The behavior of an object is specified by methods that operate on the state of the object. Each object is uniquely identified by a system-defined identifier (OID) .

Objects with the same properties and behavior are grouped into classes. An object can be an instance of only one class or an instance of several classes. Classes are organized in class hierarchies . A subclass inherits properties and methods from a superclass, and in addition, a subclass may have specific properties and methods . In some systems, a class may have more than one superclass (multiple inheritance), while in others it is ' restricted to only one superclass (single inheritance). Most models allow for overriding inherited properties and methods. Overriding is the substitution of the property domain with a new domain or the substitution of a method implementation with a different one (Chirkova2003)[17] . Achievements of the object oriented model are following:

- OODBs allow users to define abstractions
- OODBs facilitate development of some relationships
- OODBs eliminate need for user defined keys
- Development of equality predicates
- OODBs reduce need for Joins
- Performance gain using OODBs

Besides the achievements of the OODBS there is still some weakness of the OODB, like, Minimal query optimization, Lack of standard query algebra, Lack of query facilities, Limited support for consistency constraints in OODB, Little support for complex objects. The main weakness of the OODB is that it does not provide support for views. Although there have been several proposal there is little agreement as to how a view mechanism should operate in OODBs (Mumick 1997)[3]. The development of an object-oriented view capability is complicated by such model features as object identity. What are the identities of the objects in a view? On the other hand, there has also been the argument that data encapsulation and inheritance make explicit view definitions unnecessary.

3.3. An Overview of ODMG Model

The ODMG is a group of vendors and interested parties that work on specifications for object database and object-relational mapping products. ODMG embodies vendor's efforts to standardize ODMBS. The standardization of database schemas, programming languages bindings and query language open the way for portable applications. The powerful ODMG effort has given the object database industry a jump start toward standards that would otherwise have taken many years. ODMG enables many vendors to support and endorse a common object database interface to which customers write their applications. (Chirkova 2003)[17] .

The programmer writes declaration for the application schema using either a PL (Programming Language) or ODL and provides a source program to implement the application (Urbano 2003)[2]. The source program is written in one of the supported PLs using special libraries to provide object manipulation language capabilities. Source programs can be written in other manipulation language to access and manipulate the same database. The declarations and source program are then complied and linked with the runtime libraries of an ODMG compliant ODBMS to produce the running application. The major components of the ODMG standard are the OM , ODL, OQL and programming language bindings.

3.3.1. OBJECT MODEL

Refers to the collection of concepts used to describe objects, in a particular object-oriented language, specification, or analysis and design methodology, and corresponds closely to the use of the term *data model* in the relational data model (Urbano 2003)[2] .The common data model supported by ODMG implementations' is based on the OMG Object Model.

The OMG core model was designed to be a common denominator for object request brokers, object database

systems, object programming languages, and other applications. In keeping with the OMG Architecture, a profile has been designed for their model, adding components (e.g., relationships) to the OMG core object model to support the ODMG needs .

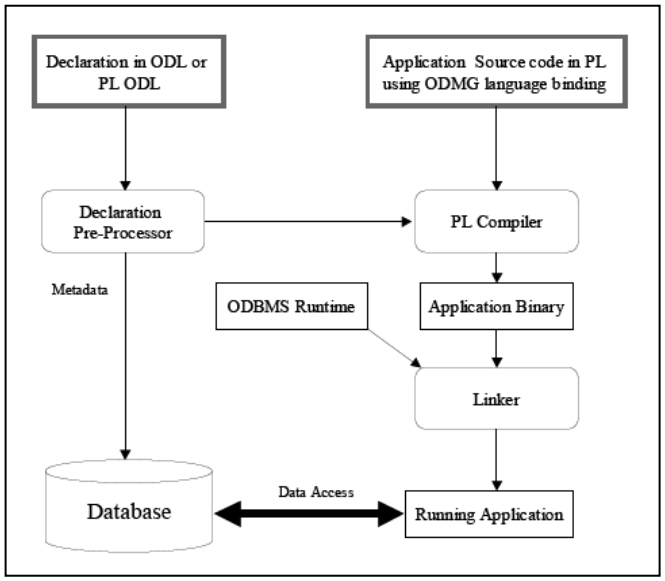

Fig 2.ODBMS Architecture

The basic concepts are objects, types, operations, properties, identity and sub typing. **Objects:** "Objects are instances of a type, and as such have state, behavior and identity. All objects are of type Denotable Object . An object can be mutable (instance of type "object") or immutable (instance of type "literal").Objects and literals can be atomic or structured. The identity of objects is represented by OIDs; typically literals are identified by their value . **Properties**: State is modeled by the properties of an object. A property can be an attribute or a relationship. The attributes and relationships of an object are defined as part of the type interface. Attributes take literals as their values; relationships can only be defined between two non literal' object types. **Operations:** Operations are defined on types. The interface of a type includes operation signatures: argument names and types, possible exceptions, result types. The first argument of an operation is distinguished.

3.3.2. Object Definition Language

The Object Definition Language is a specification language used to define the specifications of object types that conform to the ODMG Object Model. ODL is used to support the portability of database schemas across conforming ODBMSs (Urbano 2003)[2] .

Several principles have guided the development of the ODL, including:

- ODL should support all semantic constructs of the ODMG Object Model .
- ODL should not be a full programming language, but rather a definition language for object specifications .
- ODL should be programming-language independent .
- ODL should be compatible with the OMG's Interface Definition Language (IDL) .
- ODL should be extensible, not only for future functionality, but also for physical optimizations .
- ODL should be practical, providing value to application developers, while being supportable by the ODBMS vendors within a relatively short time frame after publication of the specification .

ODL is intended to define object types that can be implemented in a variety of programming languages. Therefore, ODL is not tied to the syntax of a particular programming language. Users can use ODL to define schema semantics in a programming language independent .

way. A schema specified in ODL can be supported by any ODMG-compliant ODBMS and by mixed-language implementations. ODL provides a degree of insulation for applications against the variations in both programming languages and underlying ODBMS products. Example of ODL is:

```
class Person
(extent people) {
private:
   Attribute String name;
   Attribute Set<Person> spouse;
   Attribute Set <Person> children;
   Attribute List <Person> parents;
public:
   Person(char *name);
   void birth(Person child);
   void marriage(Person spouse);
   String get_name() { return name;}
   List<Person> get_children() {return children;};
};
```

3.3.3. Object Query Language

This is a declarative (nonprocedural) language for querying and updating objects. It can be used in two different ways either as an embedded function in a programming language or as an ad-hoc query language (Urbano 2003)[2]. Object Query Language, is Declarative query language (SQL-like). It can be optimized Syntax based on SQL. Queries can return a collection of objects, an object, a collection of literals, a literal**.** The following are the some features of OQL :

- OQL is closed and complete under ODMG object model
- It can be used interactively as well as embedded in other programming languages
- OQL is not computationally complete
- OQL can be invoked from within programming languages for which an ODMG binding is defined. In addition though often not in practice OQL can invoke operations programmed in these languages .
- Although OQL does not provides explicit update operators. It can invoke operations on object that may cause updates .
- Being functional language operators in language can be freely composed. OQL includes arithmetical, logical, aggregation, sorting, and grouping operators. In addition it also provides operators for creating collections .

Example of OQL is: List the name and address of Guests with reservations for more than one day.

```
select struct(x.GuestName, x.StreetNr, x.City)
from x in Guest, y in x.has_Reservation
where y.NrDays > 1
```

Is there a reservation for the Kennedy room on 13 May?

```
exists x in Reservation : x.ArrivalDate = "13 May"
```

and x.is_for_Room.RoomName = "Kennedy"

For each room, list the cities and arrival dates of guests with reservations.

```
select struct(x.RoomName,
    (select struct(y.ArrivalDate, y.is_for_Guest.City
  from y in x.is_reserved_in_Reservation))
 from x in Room
```

Example 2 of OQL query

```
select struct (E: e.name, D:d.name)
from e in Employees, d in Departments
where e.dept = d;
define workFor() as
select struct (E: e.name, D:d.name)
from e in Employees, d in Departments
where e.dept = d;
```

3.3.4. Programming Language Binding

A languages binding is an integration of the ODMG, OM, OML, ODL and OQL with an OOPL. An OOPL together the ODMG binding provides full fledged database programming languages (Urbano 2003)[2].

The application access the database creates objects retrieves objects, manipulation objects starts transaction commits or rollback transaction. The language bindings for different OOPLS are quite different from each other in their style and strengths but they serve one common purpose to provide an OOPL with database capabilities.

3.4. Difference of OODB with Relational Model

Those aspects of the OODB standard are compared with its relational counterpart that influence the way materialized view can be incrementally maintained. The reason for such a comparison is to identity key difference between the IVM problem in the context of ODBs and RDBs and to distinguish work from comparable work on materialized view maintenance in RDBs (Ali 2000)[1].

In ODBs and OODB standard objects are uniquely identified through OIDs. While in RDBs each tuples in a relation is uniquely identified using a set of attributes values known as the primary key. Any modification to the values of such attributes change identify of the tuples. In contrast an OID is system generated and is independent of the stare of object.

Through OID and primary key serve a simpler purpose, they are fundamentally different and should not be treated as a same thing. It is possible in MV in RDBMs to give the same primary key as the relation from which it is derived. It is not possible in OODB context to create new objects with the same OIDs as existing objects which means that objects in the MV might have the same stare as that of the base objects but different identities.

In OODB Om relationships are implemented using reference objects. Relationships are bi-directional and could be collection valued. In contrast relationships in RDBs are implemented using foreign keys and are uni-directional and single valued. Bi-directionality of relationship makes IVM more difficult because of the derived updated.

There are many different Updates operations in OODB OM where they are only three types of updates in relational model that is insert, delete, and update. One update operation might have several updates associated with it. The RDBS an update affects only one relation with the exception when updates are cascaded due to referential integrity enforcement. Such updates can be seen as derived updated operations. However the difference is that in RDBS one can choose not to cascade update whereas in ODBs referential integrity is by default maintained by the ODMBS.

In RDBS when an update is applied to a relation that might have implications on other relations and that might have implication on other relations and that cascading of updates is switched off one can explicitly carry out the derived updated. In contrast in ODBs possibilities of derived updated are carried out by the ODMBS implicitly. In RDBs when a update is applied to a relation that affect an MV then an incremental change is usually computed by substituting in an algebraic expression or an SQL view definition a relation that is affected by the update with delta. This is mainly because an update affects only one relation and every algebraic operator takes one or two relations an input and replacing one relation with another one is straightforward. The OODB context this is not always possible because the delta might not be a direct input in the algebraic expression or an OQL view definition.

For example assume that an MV v is based on extents x and y such that **v:= x p y** where p is the join predicate. In an RDB v can be affected by an update that aspects the tuples of either x or y. suppose that an update u affects x then an incremental change to v is composed by delta **v := delta x p y** where delta x is the delta associates with the u. in the OODB settings an update u might affect a class extent z which implicitly affects the objects in x and hence affects v. the incremental change to v in this case may not be computed by **delta v:+ delta z p y** as x and z could be extents of entirely different classes. That is why it man not be possible to substitute in the expression defining v, x by delta z thick makes IVM more different in ODBs. The level of difficulty depends on how update events are detected. If derived updates raise events which can be responded to then there is no problem. In RDB once an incremental change is competed there are only there possible ways in which it can be applied to the MVs wither by inset delete to update.

The change itself is generally a collection of tuples that can be directly applied to the MVs. In contrast there are many different ways of applying changes to the MVs as there are update operation in OODB OM. The change itself is not necessarily directly applicable to the MVS and much be post-processed before it can applied to the MVS for example to construct new objects form the computed change or to retrieve objects from the MVs that are going to be either deleted op updated.

Table 3. Difference in RDB and ODB

Criteria	RDBMS	ODBMS
Support for object-oriented features	Does not support; It is difficult to map program object to the database	Supports extensively
Usage	Easy to use	OK for programmers; some SQL access for end users
Support for complex relationships	Does not support abstract data types	Supports a wide variety of data types and data with

		complex inter-relationships
Performance	Very good performance	Relatively less performance
Product maturity	Relatively old and so very mature	This concept is few years old and so relatively mature
The use of SQL	Extensive supports SQL	OQL is similar to SQL, but with additional features like Complex objects and object-oriented features.
Advantages	Its dependence on SQL, relatively simple query optimization hence good performance	It can handle all types of complex applications, reusability of code, less coding
Disadvantages	Inability to handle complex applications	Low performance due to complex query optimization, inability to support large-scale systems
Support from vendors	It is considered to be highly successful so the market size is very large but many vendors are moving towards ORDBMS	Presently lacking vendor support due to vast size of RDBMS market

3.5. Attributes of View Materialization in Object Oriented Paradigm

The first dimension is the definition language dimension. Solutions differ depending on the types over which views can be defined. The richer the type system the more varied the semantic issues arising in a context in which restoring consistency is the main objective. A non-exhaustive list of values in this dimension, partially ordered from least to most expressive, could be as follows: [first-normal-form (1NF) relational, nested relational, object-based]. Historically, most of the work on the IVM problem has been carried out in a 1NF relational setting. The object based case is significantly more complex and has been considered much less often in the IVM literature (Urbano 2003)[2] .

The second dimension is the manipulation language dimension. Solutions differ depending on the update events that can be responded to. The richer the data manipulation language the more varied the semantic issues arising. The type system that the solution applies to determines the primitive operations from which one defines the observable behavior of the type instances. Values in this dimension could be subsets of the following non-exhaustive set: {insert, delete, modify, implicit consequences}.This list might be extended by consideration, of operations on collections of instances, as needed in the ODMG setting and as supported by VMOP. The object-based case is significantly more complex and has been considered much less often in the MVM literature.

The third dimension is the view language dimension. Solutions differ depending on the queries that can be bound to view names. The richer the view definition language the more varied the semantic issues arising. (Urbano 2003)[2]. Incrementally maintaining views involving aggregation, negation or recursion has proved quite challenging. Values in this dimension could be sub-lists of the following non-exhaustive list of values, partially ordered from least to most expressive, could be as follows: [select-project-join (SPJ), unnest (l), nest (C), sub-queries (SQ), aggregation (I), duplicates (D), union ([), intersection (\) difference ())]. The object-based case in the ODMG setting is arguably not the most complex as far as the expressiveness of the view definition language is concerned: the deductive case is, and has been considered more often in the IVM literature .

The fourth dimension is the event processing dimension. In addition to differences related to the database languages that underlie them, solutions will also differ regarding the strategy they adopt for processing update events. One possibility is to react to the update event immediately. An alternative is to defer the reaction to the update event, typically until the end of the transaction in which the event took place . Thus, a non-exhaustive list of values in this dimension, often taken as alternatives to one another, could be as follows: [immediate, deferred].

The fifth, and final, dimension is the environmental information dimension. The final dimension used to characterize the solution space for the IVM problem concerns how much data must be available in the environment for the solution to work. Intuitively, the less data that is needed the better. Values in this dimension could be sub-lists of the following non-exhaustive list of values partially (and informally) ordered from least to most expressive: [materialized view extent, update delta, view definitions, base relations, and auxiliary views]. There is a trade-off between the amount of data required and the update events that can be accommodated by each solution. Relying on more data allows more update events to be handled .

3.6. Materialization in VMOP

A solution to the problem of incrementally maintaining materialized OQL views defined with respect to any update operation in the ODMG language bindings. In order to achieve the goal of incremental maintainability for all 'update operations, the availability of both the references to the base objects that contribute data to the MV and of the base extents required for materializing. (Urbano 2003)[2]. It is also assumed the VMOP solution is valid for MVs that refer to any ODL-definable type (Ali 2000)[1]. The VMOP solution is valid for any update operation in the ODMG standard . In terms of practicality, the VMOP solution yields incremental maintenance plans (IMPs) at the algebraic level. This makes it easier to integrate the VMOP solution into the kind of query processing frameworks that mainstream DBMSs rely on . These include derived events that arise in the ODMG model as a result of the requirement to enforce referential integrity constraints declared with the relationship/inverse construction. Depending on the event type, one or more algebraic IMPs are constructed that, when evaluated, compute the required changes to the MV.

A crucial component in the VMOP solution is, therefore, the generation of IMPs that are appropriate for each kind of update event . In VMOP, two kinds of IMP suffice to

compute the changes required in the MV as a result of any update operation in the ODMG standard. Immediately after an update event takes place which implies the need to update an MV, the corresponding delta (comprising the old and the new state of affected objects) is made available and the associated IMP (which uses the delta) is evaluated to compute the changes needed. Once the changes needed have been obtained, VMOP applies them to the relevant MV extent .

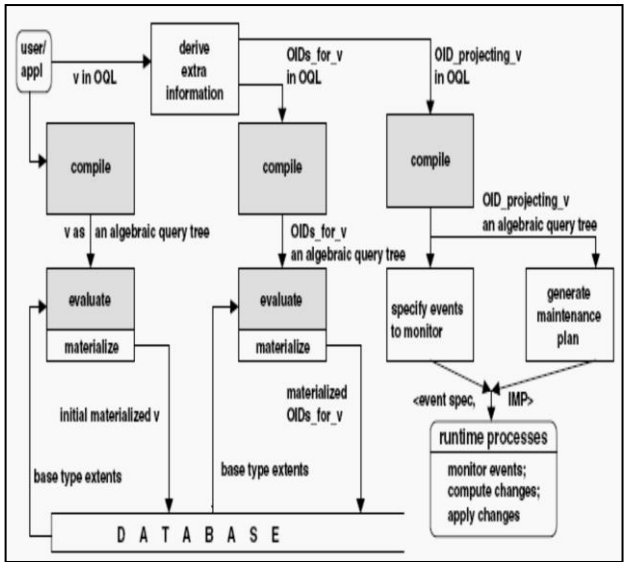

Fig 3.View Compilation

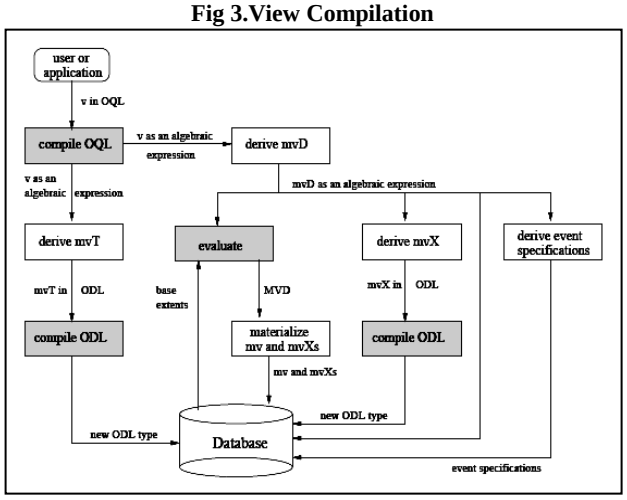

Fig 4.View Materialization in VMOP

The comprehensive nature of the VMOP solution with respect to update operations requires that the OIDs of objects that contribute data to the MV are also materialized . This is achieved, at view compilation time, by generating from an MV definition v another view definition, which is referred to as OIDs_for_v, which is itself compiled, evaluated and materialized. Thus, the OIDs of objects that contribute data for an instance of v are associated withthe OID of that instance in OIDs_for_v .

3.7. Incremental maintenance in VMOP

In VMOP, the generation of an IMP starts from an algebraic query tree in which denotations are available for the OIDs of contributing objects which is referred to as OID_projecting_v. It is identical to v except that it also includes those attributes in OIDs_for_v that originate from extents occurring in the from clause of v. In contrast with v and OIDs_for_v, OID_projecting_v is never materialized: only the definition is needed as an input to the generation of the IMPs .

After extra information is derived, the events to be monitored are identified and their corresponding IMPs generated. Different forms of IMPs are generated depending on the update and the properties of the view, as follows.

Planting a delta For some kinds of updates, the constructed IMP computes the changes required to v by evaluating OID_projecting_v over the delta to the affected base extent, rather than over the base extent itself, while accessing all other base extents refeenced in the MV. For insertions only the new state matters for incremental MV maintenance. Correspondingly, for deletions only the old state matters. However, for modification, both the old and new states are necessary, so the representation of the delta benefits from being conceptualized as a pair. The IMP generated by VMOP is different from the evaluation plan. **Joining a delta with materialized OIDs.** "For other kinds of updates, the IMP constructed by the system joins OIDs_for_v with the delta in order to identify MV objects that are affected by the update. The idea is to avoid access to base extents whenever possible. The information captured in the delta is not enough to identify which object(s) in the MV might be affected because the delta only refers to the updated composite Part object: there is no handle in the delta to the instances in the MV which have data that was contributed by the affected composite Part object. In this case, the IMP will need to join OIDs_for_dbSizeView with the delta on the OID of the object in the delta. When this IMP is evaluated, the result is input to an algorithm that applies the changes to the MV and to OIDs_ for_dbSizeView, if required.

3.8. Algorithm for View Materialization in Object Oriented Paradigm

VMOP processes that are carried out when update events take place. For explanation, given a database state D over a schema S that conforms to ODMG. Suppose that an update u E ET causes a transition from D to D'. Then the following action takes place:

1 Generating an event and delta: from the update u an event e and a delta(new, old) are generated and passed on to subsequent steps.

2 detecting relevant event: the event e is matched against the event specifications. A relevant event and corresponding event specifications are passed on to subsequent steps.

3 Processing relevant events: for e and each corresponding event specifications es there exists one or more v from which es was derived. For each affected v, following action take place:

- Identifying the type of incremental plan
- Generating the incremental plan
- Applying updates to materialized views

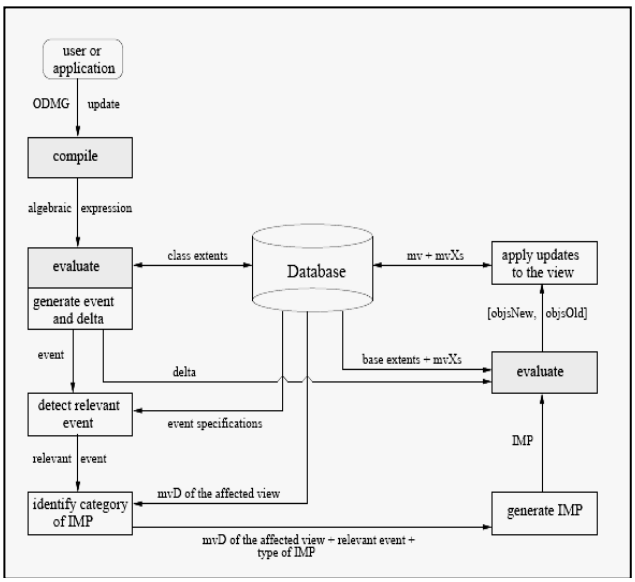

Fig5.Incremental Maintenance in VMOP

3.9. Type of Incremental Maintenance Plan

Choosing an IMP type involves the following analysis:
Kind of update event: updates events can be divided into two groups. first updates that cause new objects or elements to be added to the view, i.e. insert and insertElem and second updates that cause removal or modification of existing view object, i.e. delete, deleteElem and modify (Urbano 2003).
view classification: views can be divided into four classes depending on the operators that occur in their algebraic representation in increasing levels of complexity. These are reduceget, reducegetjoin, unnest and nest (Urbano 2003). Given the kind of update event and the views class the IMP types is determined. There are two types of IMP

- IMPs that requires access to base extents. It is referred to as planting a delta in mvD.
- IMP the do not require access to base extents. It is referred to as joining a delta with mvXs.

3.10. Generating an Incremental plans

For each type of incremental maintenance plan, VMOP uses a different algorithm to generate an IMP. Therefore there are different algorithms used for each plan. The key points in the incremental maintenance plan are as follows:

- Base extents are not accessed.
- The IMP joins the delta with the extra information. The extent of the materialized extra information shows the join condition. The join condition matches the objects in the delta with the objects in newer state by comparing the OID of the former with the CompositePart_ID attribute of the latter which hold the OID of a compositePart object that was contributed data to the view.
- The IMP returns a structure containing the OIDs of the view object and the new value of the CompositeType attributed of the view.

3.11. Applying Updates to Materialized View

Once the Imps are generated there are evaluated to obtain incremental changes to MV. It is then necessary to apply the incremental changes to MVs to make them consistent with the current state of the database. The specific way in which changes are applied to MVs depends on the kind of update event and the view class.

3.12. Propagating Changes to the Views

Given a materialized view v that is affected by an event e the set of changes obtained by evaluating an Imp that used delta new is denoted BY. If imp uses delta old then the set of changes is denoted BY. The explanation is given as:

1. View update Operations : The view updates operations that are used in VMOP for applying changes to an MV. For a given view v. mvT and mv denote the ODL type and extent of v respectively mvX and mvXs denote the OLD type and extent of extra information pertaining to v. and mvD denote the augmented form of v. let V, X and D the set of attributed and collection names in mvT, mvXs and mvD respectively used the function defined as follows:

- Create an object of type t with state s where s is a list of values. The state of an object means that it is the result og appending to s2 to s1. Each element in s corresponds to an attribute in t.
- State returns the state of the object o.
- OID returns the OID of the object o.
- Position returns the position of an element el in the list collection c.
- Projection returns projection of the values of L a list of attributes and collection names, out of an object o.

The order of the object maintained the order of objects in mv. The order of mv may be affected by any of the following:

- Insertion into the base extent
- Deletion from the base extent
- Modifications to e affecting exp

The order of the mv is denoted by exp, VMOP must include at view compilation time e in v as projected attributed if they are not already included. Some of the approaches to help maintain the order of the view include:

- Adding an extra attributes index in c that will store that positron of an object in mv. A single insertion into or deletion from mv may affect the value of the index of several objects in mv. Therefore maintaining index valued could be an expensive task.
- Leaving mv unordered and providing a interface to the view such that any retrieval from mv returns the objects in the required order. A simple interface might be to define a virtual view over mv that retrieves the objects over mv.
- To store mv as a list collection. Any insertion into or deletion from mv can be carried out using appropriate positions relative to the value of exp. Modification to exp may be complicated to handle as they may require moving objects from one position to another.
- To create an index over mv on exp and let the underlying ODMBS maintain the index.

2 Applying changes to VMOP: The result given by evaluating the IMP chosen and generated views are then proceeds to apply changes to MV using the view update operator**.**

3.13. Implementation stage

Lambda DB is chosen as a platform for implementing the VMOP solution due to the following features:

- It is based on a theoretical framework called monoid comprehension calculus for query optimization
- It is based on the ODMG standard which coincides with the object based setting on which VMOP solution stands.
- It supports most of the ODL which is appropriate for realizing the data model dimension of the IVM solution space that VMOP targets.
- It supports most OQL query constructs which is more than enough to realize the VDL dimension of the IVM solution space that VMOP targets.
- It supports function for manipulating data dictionary which are helpful for realizing algorithms.
- It supports an object algebra based on the monoid comprehensive calculus which is capable of representing queries and views in OQL. The algorithms for generating Imp manipulate expression in object algebra.
- The lambda DB OQL optimizer is written in an optimizer specification language called OPTL and is implemented in an optimizer generation tool called OPTGEN. OPTL is a language for specifying query optimizers that captures most of the optimizer specification information in a non procedural style.
- Its source code is publicly available so it is possible for us to implement VMOP as an extension to it and have significant control over the overall functionality of the system.

3.13.1. Handling View materialization

View definition process is responsible for deriving and storing event specifications and metadata from MVs and generating code for materializing the initial extents of MVs. Two passes are required when view definitions are processed for materialization in the following manner:

- In the first pass, the OLD types for each view v are generated which is processed by ODL to SDL translator to plant the new schema into the data dictionary.
- In the second pass event specification and metadata are derived and stored for ivm_metadata_catalog foe each view.

The process generates code to materialize view implements the procedure by generating code that when run materializes the view and extra information. After deriving the set of event specifications the process iterates over the collection and constructs EventSpec objects which are persistently stored in the data dictionary. It also constructs an object of type MatView for the metadata pertaining to the view.

3.13.2. Handling View Maintenance

Handling View maintenance describes how view maintenance has been implemented in the VMOP. View maintenance is handled through the following way:

Generating Events from lambda-DB updates: lambda-DB does not support the ODMG C++ language binding because it provides its own C++ binding which integrates OQL with C++ called lambda-OQL. In OQL any statement that starts with thee symbol % denotes an OQL query , view definition or an update. lambda-OQL have their semantics that are similar to update operations defined by ODMG OM. As an example consider the following lambda-OQL update:

%ComPart501:=CompositePart(id:501,type:"type003",build Date1999);

From this update following event is generated:

<eventType: insert,

Params:[<classExtent, "COmpositePart", "CompositeParts">, <object, "CompositePart","CompositePart501">]>

Computing and propagating Changes to MVs:

For each lambda_OQL update event is generated, whenever there is an MV for which the event is relevant because it matches the event specifications derived for that MV at view compilation time the generated IVM code is such that when executed it evaluated the IMO associated with that event to compute the changes and then propagates those changes to MV. For each event update t, event e and associated event specification es, the IVM code is generated in the following ways:

- The process generates code to generate delta constructs an algebraic expression denoted by q and passes it to lambda_DB-QOT to generate the code foe evaluating q. q corresponding to an OQL query of the form Select x from x in the X where p.

That returns a collection containing the objects affected by the event where X is name of the extent of the affected objects and p is a predicate. Both X and p is derived from the information carried by t and e.

- For each MatView mv associated with es , following action take place:
- An algorithm is implemented for choosing an appropriate IMO type and generating an IMP. The process generates an Imp denoted by imp and returns it to the calling process.
- Manage code invoke the process with the input imp and mv. The generation of the IVM code that when executed computes the changes are necessary to be applied to the MV.
- All generated code is serialized and written out to the source code files. These files are compiled and linked to obtain an executable that when run computes and propagates changes to the affected MVs.

Table 4. VMOP in IVM solution space

Dimension	Position
Data Model	$ODMG_{TP}$ (excluding bag and list literal collections)
View Definition Language	OQL_F (excluding views with `distinct` and `order by`)
Data Manipulation Language	ET (excluding derived events and events that modify a CA)
Information Availability	$INFO$
Maintenance Timing	MT_I

3.14. Performance Evaluation

Performance evaluation can be characterized on the basis of cost modal and on the basis of experiments. A systematic approach to performance evaluation can be as follows:

- Describe the goals and the system that is being analyses.
- List the services and outcomes of the system.
- Choose the metrics and parameters used in the evaluation.
- Identify the factors to study.
- Select an evaluation technique.
- Design and conduct experiments.
- Analyses and present results.

The factors that influence the performance of an IVM system are studied. The evaluation technique used in the PE is

empirical in nature based on measurement of various metrics using the VMOP prototype. The choice of which views to use for the evaluation is based on the hypotheses being investigated in the performance analysis.

- The selectivity of the view influence the performance of an IVM as the ratio of the number of objects selected by a query to the number of inputs objects .
- The structural complexity of the view influence the performance of an IVM system it varies with the number and kind of algebraic operators needed to evaluate the query part or the view .
- The benefits of an IVM system increase with an increase in database size. The number of module objects varies uniformly across the databases. The effects of databases size on the performance on an IVM system relative to its counterparts .

The performance of an IVM system can be evaluated by:

- The cost of query answering over the MV and compared with its virtual counterpart.
- The cost of incrementally maintaining the MV by update propagation compared with re-materialization .
- The cost of incrementally maintain the MV in comparison with the cost of answering a query over its virtual counterpart .

Experiments are carried out to testing the hypothesis. Each experiment investigates the above three kinds of cost. These experiments were run in cold state so that conduction one experiments does not favor or discriminate another one due to the object caching used by the underlying DBMS. Each experiment was run three times and the average taken. The elapsed time for answering a query or propagating an update is measured in milliseconds.

6. FUTURE WORK

Currently VMOP does not support array and dictionary collection types. Supporting these collections would complete the entire ODMG OM. In order to support all update operation defined on the ODMG OM it would require extending the MOIE solution algorithms for deriving event specifications , generating IMP and applying changes to MV to support update operations specific to array and dictionary collection types.

MV can potentially provide improvements in query processing times and will be useful if query optimizers can use MV transparently in order to answer queries. In order to realize this in VMOP the OQL optimizer must know how and when to exploit MV. The optimizer should be able to determine whether part or all of a query can be computed from MVs. The performance of VMOP does not result in a generic policy for selecting views for materialization. This may also be a research area so that system can identify and choose views that are likely to be beneficial.

REFERENCES

[1] Ali M.A. and N.W.Paton, 2000, Incremental Maintenance of Materialized OQL views, Proceedings of the 3rd ACM international workshop on Data warehousing and OLAP, Pages: 41-48.

[2] Urbano R., 2003, Oracle Database Advance Replication.

[3] Mumick I.S., D.Quass and B.S.Mumick,1997, Maintenance of Data Cubes and Summary Tables in a Warehouse.

[4] Rundensteiner E.A. and H.A.Kuno , 1996, Using Object-Oriented Principles to Optimize Update Propagation to Materialized Views, ICDE Proceedings of the Twelfth International Conference on Data Engineering, IEEE Computer Society, Pages: 310 – 317.

[5] Vista D., 1996, Optimizing Incremental View Maintenance Expressions in Relational Databases.

[6] Bebel B. and R.Wrembel, 2001, Hierarchical Materialization of Methods in Object-Oriented Views: Design, Maintenance, and Experimental Evaluation, Atlanta, Georgia, USA.

[7] Mistry H., P.Roy, S.Subsarshan and K.Ramamritham, 2001, Materialized View Selection and Maintenance Using Multi-Query Optimization, SIGMOD Conference, pp. 307–318.

[8] Encinas S., M.Trinidad, H.Montano and J.Antonio, 2007, Algorithm for Selection of Materialized Views: Based on Costs Model, Current Trends in Computer Science, Eighth Mexican International Conference on Digital Object Identifier, Issue 24-28, Pages: 18 – 24.

[9] Yang J., K.Karlapalem and Q.Li, 1997, Algorithms for Materialized View Design in Data Warehousing Environment, Issue 136-145.

[10] Rundensteiner E.A. and H.A.Kuno , 1996, Using Object-Oriented Principles to Optimize Update Propagation to Materialized Views, ICDE Proceedings of the Twelfth International Conference on Data Engineering, IEEE Computer Society, Pages: 310 – 317.

[11] Gupta A. and I.S.Mumick, 1999, Materialized Views Techniques, Implementation and Applications.

[12] Roy P, S.Seshadri, S.Sudarshan and S.Bhobe,2000, Efficient and Extensible Algorithms for Multi Query Optimization, ACM.

[13] Goldstein J. and P.A.Larson, 2001,Optimizing Queries Using Materialized Views: A Practical, Scalable Solution, ACM SIGMOD.

[14] Lee Y.K., J.H.Son, and M.H.Kim, 2001,Efficient Incremental View Maintenance In Data Warehouses ,Conference on Information and Knowledge Management Proceedings of the tenth international conference on Information and knowledgemanagement,Pages:349-356.

[15] Palpanas T., R.Sidle, R.Cochrance and H.Pirahesh, 2002, Incremental Maintenance for Non-distributive Aggregate Functions, Very Large Data Bases, Proceedings of the 28th international conference on Very Large Data Bases, Pages: 802 – 813.

[16] Ali M.A. and N.W.Paton, 2003, MOVIE: An Incremental Maintenance System for Materialized Object Views.

[17] Chirkova R. and C.Li, 2003, Materializing Views with Minimal Size to Answer Queries, ACM.

[18] Paraboschi S. G.sindoni, E.Baralis and E.Teniente, 2003, Materialized Views in Muiltidemensional Databases,

Multidemensional Databases: Problems and Solutions,Pages: 222-251.

[19] Lee Y.K., J.H.Son, and M.H.Kim, 2005, Optimizing the Incremental Maintenance of Multiple Join, Views, Data Warehousing, and OLAP, Proceedings of the 8th ACM international workshop on Data warehousing and OLAP, Pages: 107 - 113.

[20] Gupta H. and I.S.Mumick, 2006, Incremental maintenance of aggregate and outerjoin expressions, Information Systems Volume 31, Issue 6, Pages: 435 – 464.

[21] Dhote C. and M. S. Ali, 2007, Materialized View Selection in Data Warehousing, International Conference on Information Technology, IEEE Computer Society Washington, DC, USA.

[22] Hung M.C., M.L.Hunang, D.L.Yang and N.L.Hsueh ,2007, Efficient Approaches for Materialized Views Selection in a Data Warehouse, Information Sciences: An International Journal, Volume 177, issue 6, Pages: 1333-1348

[23] Yin G., X.Yu and L.Lin , 2007, Strategy of Selecting Materialized Views Based on Cache updating, IEEE, International Conference on Integration Technology.

[24] Zhou J., 2007, Lazy Maintenance of Materialized Views.

[25] Mr. P. P. Karde, and Dr. V. M. Thakare,2010, Selection Of Materialized View Using Query Optimization In Database Management, International Journal of Computer Science & Engineering Survey (IJCSES)

[26] Mr. P. P. Karde, Dr. V. M. Thakare,2010, Selection & Maintenance of Materialized View and It's Application for Fast Query Processing, *International Journal of Computer Science & Engineering Survey (IJCSES)*

[27] Prabagaran G., N. Venkatesan, 2013, Issues of Materialized Views in Dataware housing for Efficient Management of Information , *2nd National Conference on Future Computing February 2013*

[28] Anjana G., S. Sabharwal and R. Gupta, 2015, Model Based Materialized View Evolution, *Science Direct*, 1273-1280.

[29] Colby L.S., 1996, Algorithms for Deferred View Maintenance, ACM SIGMOD Record Volume 25, Issue 2, Pages: 469 – 480