Query Optimization in OODBMS: Decomposition of Query and cached for Wider Query Management

Ms. S.S. Dhande
Asso. Prof
Computer Science & Engineering
Sipna's College of Engg & Tech,
Amravati, Maharashtra,
India.
sheetaldhandedandge@gmail.com

Dr. G. R. Bamnote
Prof & HOD
Computer Science &Engineering
Prof. Ram Meghe. Institute of Research , Badnera
Amravati, Maharashtra,
India.
grbamnote@rediffmail.com

ABSTRACT

This paper is based on relatively newer approach for query optimization in object databases, which uses query decomposition and cached query results to improve execution a query. Issues that are focused here is fast retrieval and high reuse of cached queries, Decompose Query into Sub query, Decomposition of complex queries into smaller for fast retrieval of result.

Here we try to address another open area of query caching like handling wider queries. By using some parts of cached results helpful for answering other queries (wider Queries) and combining many cached queries while producing the result.

Multiple experiments were performed to prove the productivity of this newer way of optimizing a query. The limitation of this technique is that it's useful especially in scenarios where data manipulation rate is very low as compared to data retrieval rate.

Keywords— Query Caching, Query Decomposition, Query Optimization, Stack Based Approach (SBA), Stack-Based Query Language (SBQL), Object Databases, Query Performance, Query Evaluation.

1. INTRODUCTION

In various types of Database Systems (Relational as well as Object-Oriented), many techniques for query optimization are available [1]. Few of them are Pipelining, Parallel Execution, Partitioning, Indexes, Materialized Views, and Hints etc [2][4].

One technique which has not been convincingly implemented is Query Caching [3][20].

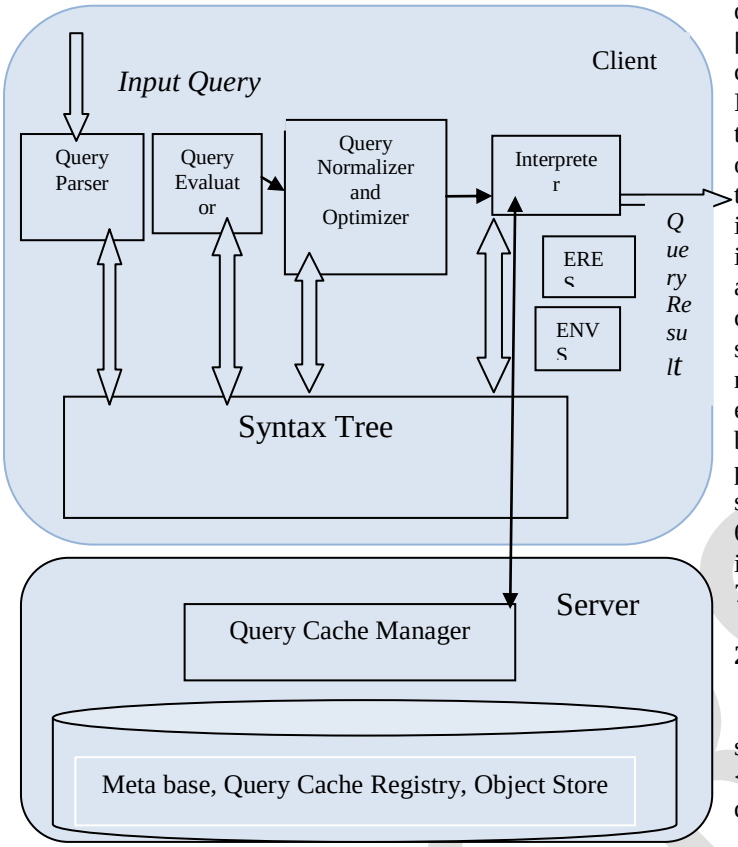
Query Caching will provide optimum performance. Instead of spending time re-evaluating the query, the database can directly fetch the results from already stored cache. The most obvious benefit of Query Caching can be seen in systems where Data Retrieval rate is very high when compared to Data Manipulation. Hence database i.e. data store get modified after the long periodic intervals. During these intervals if a particular Query is calculated only once i.e. for the first time and 999 times the stored result is reused [3][4]. Data Manipulation can invalidate the cache results because the inserted /modified /deleted data can bring the difference between the cached results and the actual results. Hence regeneration of the cached results will be required to restore the results back again to the useful state [5][19].

Oracle 11g database system offers this mechanism for SQL and PLSQL. Mysql database also implemented query chaining where only full selected query texts together with the corresponding result stored in cache. LINQ, Microsoft .NET environment also have this kind of facility [4].

Our research is focused on how cached queries transparent mechanism can be used in query optimization, assuming no changes to syntax, semantics and pragmatics of query language itself [6]. But there is not any result caching solution implemented in current commercial and non –commercial object oriented database system [7]. The most important concept of this work, Query caching have been implemented by constructing a prototype [8][9] . Most of Commercial object databases used OQL as query language proposed as model language by ODMG [2][10][11].

The experimentation for optimization of query will be done based on queries written in Stack based query language (SBQL) syntax which is designed and implemented using a stack-based architecture (SBA) framework [12][13].The performance gains will be measured by comparing the results against the performance of the identical queries written and executed in Prototype with cached and without cached queries. This paper organized as section 1 Introduction of concept. section 2 System Design and Architecture, shortly describes description of optimization strategies-Query caching and further reuse the result in cached by earlier execution which is semantically mapped(Decomposition , Normalization) section 3 shows experimental findings and section 4 Discuss Experimental Result of cached semantically mapped queries and concludes.

2. SYSTEM DESIGN ARCHITECTURE



2.1 Query Optimization Module

The scenario of the optimization using cached queries in query evaluation environment for SBA is as follows.[14]
1) A user submits a query to a client-side user interface.
2) The user interface system passes it to the parser. The parser receives it and transforms into a syntactic tree
3) The query syntax tree is then received by static evaluator and type checker. It checks whether the query is syntactically correct or not. If not, it will report the errors. It also validates the table names, column names, operators, procedure names, function parameters involved query. Hence it will check the query's semantics. For this purpose, it will use the Metabase present on the server side. Metabase is a part of the database system which contains the Meta information related with the data in the various objects. This is static evaluation of the various nodes in the query syntax tree [14].
4) This type checked and statically evaluated query tree is then sent to the query normaliser which reconstructs the query according to the rules of normalization. This normalised query is then send to the query optimizer. All these components query parser, query type checker; query normaliser, query optimizer and query interpreter are employed on the client side database system.
5) The query optimizer rewrites the received normalized query using particular strategies like query decomposition. Each decomposed part of the complex query is send to the server [15]. Server checks whether the received sub query is already cached or not. If sub query is present in cache, the Unique Identification number of the entry in cache which corresponds to the result of the given sub query is dispatched to the query optimizer. Optimizer replaces (rewrites) the sub query tree of the total query tree by a node containing that unique identification number [16]. This UIN will be used by query interpreter to directly fetch the result from the server. Hence all the parts of the query whose results are already stored in cache will get replaced by their respective UIN. Due to this all sub queries which get replaced by corresponding UIN, their results will be brought from the cache & hence their re-evaluation will be avoided. This rewriting will generate the best evaluation plan which promises to give the best performance & having a least cost in terms of time and storage.
6) The optimized query evaluation plan is then sent to query interpreter.
7) The plan is executed by the query interpreter [16].

2.2 Query Caching

Once the optimised evaluation plan is executed successfully, the query is cached on the server side in pair <query, result>. Following that the calculated result of the query is send to the client user who has submitted the query.

When the semantically equivalent query (written in the same or different way) is submitted by the same or other user, after parsing, type checking and normalization of the query, optimizer sends the query to the server side query cache manager. Query cache manager searches for the query in the query cache registry and if found there will return the unique identification number (UIN) of the corresponding result to the client, thus avoiding the recalculation of previously stored result. Using this UIN, query interpreter (on the client system) can fetch the stored result of the query directly from the server. If the query is not found in query cache registry, query cache manager will send a message to an optimizer (on the client side database system) indicating that a query is not cached and hence its result needs to be calculated. Optimizer then does not rewrite the query i.e. does not reconstruct a parse tree. That part of the query will be then calculated by the query interpreter at runtime using runtime ENVS (Environmental Stack) and runtime QRES (Result Stack)[14] [17].
Description of few components on server side:
Query Cache Manager – This is a program running in the server, its job is to check the Query Cache Registry and figure out if the query is cached. The Query Optimizer with pass normalized query (or normalised inner sub-query) to the Query Cache Manager.

Query Cache Registry – This contains all the cached queries along with the results.

2.3 Query Normalization

To prevent from placing in the cache, queries with different textual forms but the same semantic meaning (& hence also will generate the same result), several query text normalization methods will be used. Hence if a query is already stored in the cached with its result, all semantically equivalent queries will make reuse of the stored result, as all those queries will be mapped with the already stored query (due to normalization) [9].

Examples of few techniques useful in the process of normalization are:

a) Alphabetical ordering of operands
b) Unification of Auxiliary Names
c) Ordering based on column names (in the order in which they appear in the table description).
d) Column names should be maintained to the left side of each operator [4].

Algorithm for query execution:
Step 1: Receive the query.
Step 2: Divide the query into set of tokens
Step 3: Construct the parse tree from the set of tokens
Step 4: Normalize the parse tree by applying normalization rules.
Step 5: Traverse the normalized parse tree to get normalized text query.
Step 6: Send the normalized text query to cache optimizer.
Step 7: Decompose the normalized text query into set of queries.
Step 8: While the complete result of the query is not known
Do
Send the smallest independent subquery to the query cache manager.
If
the result reference is received
Replace the subquery with the reference
Else
Calculate the result & Cache the query.
[endif]
Done

Procedure parsetree()
{
1. Find the first "WHERE". This is the root node
2. The tablename before "WHERE" should be the left child node Also mark the tablename node (MARKED).
3. Find the AND or OR if present in the query if not present goto step - 4
3a the condition after the AND or OR should be the right subtree rooted at operator node.
3b For each operator node set the value/columnname to its left as left node & right as right node.
3c goto step-5
4. The condition after where should be the right child node.
4.a Look for binary operators and create a node for them. (>, = etc)
4 b.The values/columns before and after the binary operator become the left and right node.
4 c. If there is a query after the binary operator then do the steps from 1 again. Till you reach the end.
TABLE I. *Get the list of columnnames following the).*
TABLE II. *Attach the list of columnnames to the MARKED node.*
}

Procedure Normalise ()
{
operatorslist [] = {=,!=,<=,>=,>,<};
read a query
repeat for all the operators in the query
{
// ensures constants lies to the right of operator.
if(columnvalue is to the left of operator)
swap the left & right side of the operator.
if(operator has on both sides table attributes)
{
serialize the condition on tablenames in the list of tables in the database.
}
}
Repeat for each 'and' || 'or' in the query
{
//occuranceof returns the occurance number of the operator in the operators list.
if occuranceof (left-side condition operator) > occuranceof (right-side condition operator)
swap the left and right side conditions
else if occuranceof (left-side condition operator) == occuranceof (right-side condition operator)
{
//occuranceof returns the occurance number of the column in the table description.
if (occuranceof (left-side condition columnname) > occuranceof(right-side condition columnname)
swap the left and right side conditions
}
}
For list of attributes after each).'
Rearrange the list of attributes by referring/according to table description
For I =0 to numberofauxiliarynames do
Auxiliary-name[i] = "AUX" + I;
}

Normalized Parse Tree for "employee where salary < ((employee where name = 'krishna_kant').salary)
"

To implement this prototype we have design School database. Where School, Student, Grade, Result (Grade n Result are complimentary class) classes created with associations.

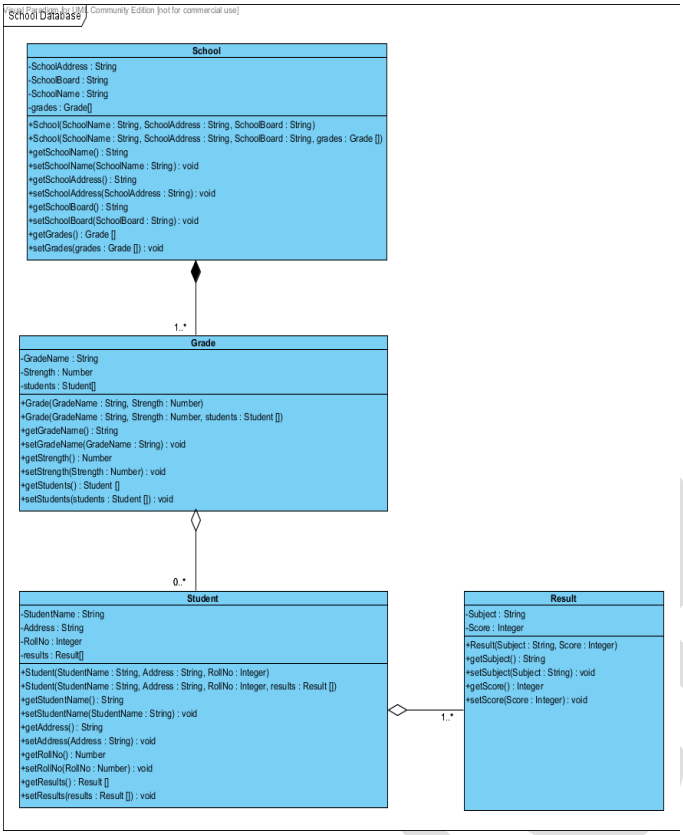


Figure 2: Class Diagram of Student Database.

2.4 Query Decomposition and Rewriting

After normalization phase query is decomposed, if possible, into one or many simpler candidate sub queries. Query decomposition is a useful mechanism to speed up evaluating a greater number of new queries. If we materialize a small independent sub query instead of a whole complex query, then the probability of reusing of its results is raised. Because the same query may occur as a sub query in many other queries, hence reuse of its stored result will speed up the performance of all those queries and as a whole of database system.

Following is the implementation of wider query Q , decompose into sub query Q1, Q2, Q3. Wider query consider here which contain many AND, OR operations (wider query some time may not a complex query but combination of many sub query). School database created with 1500 students in total 3 Schools (500 students in each) at the back-end of the simulator application in a file (by using serialization of objects) for above School schema and also created the identical database in DB4o. I have designed some queries (Query by Example/Native/Soda) in DB4o and designed equivalent queries inside the Prototype in format similar to Stack Based Query Language (SBQL). Prototype is using the caching as the optimization technique for queries (written in SBQL syntax).

Query Q: Find out Students secure more than 75 % of Marks in Examination from School "AAA" and studying in "CBSC" Board

Taking into consideration of above class diagram (Figure 2), SBQL syntax Query for above Query

Student where Score > ((Student where schoolName= "AAA") AND (Student where schoolBoard="CBSC") AND (Student where Score > 75)).StudentName

Decompose query in to three different independent queries Q1, Q2, Q3. Factoring out Independent Sub queries:

We segregate out an internal independent query: where Q1, Q2, Q3 are auxiliary name for queries.

Student where Score > ((Student where schoolName= "AAA" as Q1)

Student where Score > (Student where schoolBoard="CBSC" as Q2)

Student where Score > (Student where Score > 75 as Q3)

Now transform whole query to the following form

(Student where Score > Q1 join Q2 join Q3). StudentName

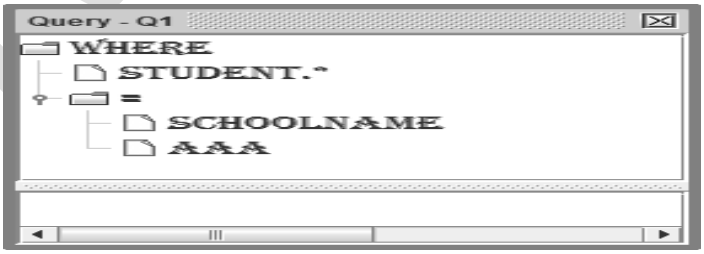


Figure 3: Normalized Parse Tree For Query Q1.

And the normalized query we get from the normalized parse tree by applying normalization on it. Hence any query which is semantically equal this further used from cache.

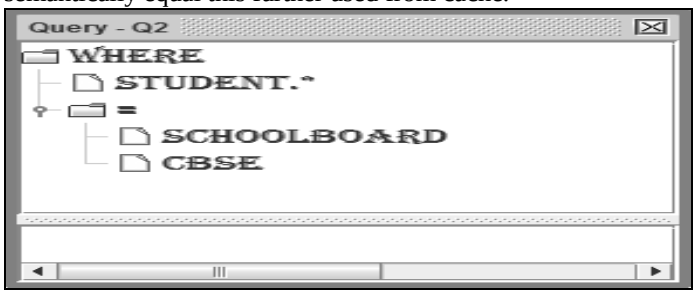


Figure 4: Normalized Parse Tree For Query Q2.

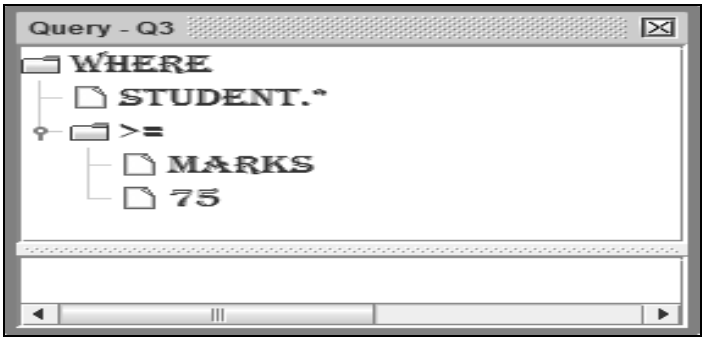


Figure 5: Normalized Parse Tree For Query Q3.

All this Q1, Q2, Q3 normalised query will store in the cache.

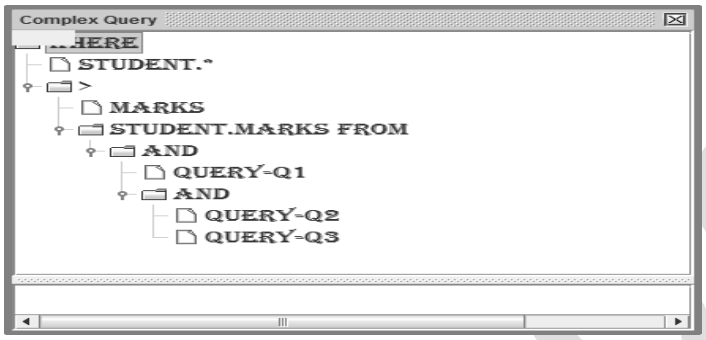


Figure 6: Normalised Parse Tree for Query Q.where cached Subquery Q1, Q2, Q3, are linked.

3. EXPERIMENTAL RESULT

We have the performance of the optimizer by calculating response times 100 subsequent results using set of queries retrieving data from database containing over 15000 objects (we have also created 3 different frames containing the same application but first frame is connected to data source containing 1500 students records, second frame is connected to data source containing 7500 students records and third frame is connected to data source with 15000 students records.) Instance of student schema presented in above section. We have compare data with four optimization strategies with cache, without cache, with Db40, and in many cases response time were 100 times faster. Here we try to show comparative result by sampling of three queries.

Approch	Query One	Query Two	Query Three
DB4o [no optimization]	150087	144779	117553
Prototype [no caching]	375127	391564	345777
Prototype[with caching]	21000	19845	20221

Note: Numbers indicates time taken to calculate the result of the query in microseconds

Table 1: Comparison of DB4o Object Database System Performance and Performance of Prototype Application for sample 3 complex queries

Queries	Data Source One	Data Source Two	Data Source Three
Query One	15093	69635	165250
Query Two	44297	89709	356978
Query Three	23006	96759	125690
Query Four	35897	84760	221977

Table2 : Queries execution time increases with the growing size of the database

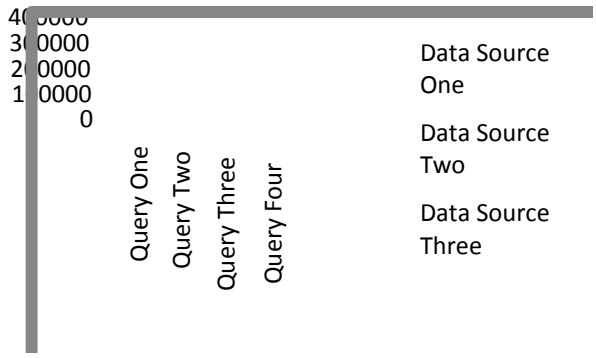


Figure 8: Time Taken for calculating result of query increases with increase in size of database.

CONCLUSION

Based on the experimental results we can state that Decomposition and Caching techniques in Object Oriented Queries have resulted in approx around hundred percent increases in performance and query output. Here we try to address future scope [] of combing many cached query results for producing result of more complex queries. High performance of these techniques will make these queries ideal for scenarios when Data Retrieval ratio is very high when compared to Data Manipulation. This is due to the fact that the

cached results will not have to update with the latest data at a frequent interval which in turn will boost the performance of the database. With more development in these techniques, the database can be a boon in the areas of Data Warehousing which work mostly in Data Retrieval mode.

REFERENCES

[1] Yannis E. Ioannidis, "Query Optimization" *in International Journal on Very Large Data Bases VLDB Journal* ,Volume 6, Issue No 2, May 1997 Springer- Verlag New York, USA.

[2] Antonio Albano, Giorgio Ghelli, and Renzo Orsini "Programming Language of Object Database" *in Very Large Data Bases VLDB Journal*, Volume 4, 1995, Pages 403-444.

[3] Hanna Kozankiewicz, Krzysztof Stencel, Kazimierz Subieta "Distributed Query Optimization in the Stack-Based Approach" *Springer Journal Lecture Notes in Computer Science*, Volume 3726,pages 904-909, Springer-Verlag Berlin Heidelberg 2005.

[4] Silberchatz ,Korth ,Sudharshan *"Database System Concepts*" 4th edition Mc-Graw-Hill, ISBN 0-07-120413-X chapter 8, Objec Oriented Database pages 307-333, chapter 13& 14 Query processing,Query Optimization,2002.

[5] Steenhagen, Hendrika Janna "Optimization of Object Query Language" *With index, ref. Subject headings: database systems / query optimization*, ISBN 90–9008745–1 ,1995.

[6] Christian Rich, Marc H. Scholl "Query Optimization in OODBMS" in Proceeding of *The German Database Conference BTW*., Springer, ISBN 3-540-56487-X March 1993.

[7] Minyar Sassi and Amel Grissa-Touzi "Contribution to the Query Optimization in the Object-Oriented Databases" *in Journal of World Academy of Science, Engineering and Technology*, WASTE Issue No. 11, 2005.

[8] S. S. Dhande,G. R. Bamnote "Query Optimization in Object oriented Database through detecting independent sub queries" *in International Journal of Advanced Research in Computer science and software Engineering. (IJARCSS)*, ISSN: 2277-128X, VOL-2,ISSUE-2 , FEB 2012.

[9] S. S. Dhande,G. R. Bamnote "Query Optimization of OODBMS: Semantic Matching of Cached Queries using Normalization*" in International Conference on "Emerging Research in Computing, Information, Communication and Applications"ERCICA-2013, Elsevier Publication DBLP,ISBN:978-93-5107-102-0, Aug 2013.*

[10] Rodrigo Machado, Alvaro Freitas Moreira, Renata de Matos Galante "Type-Checking OQL Queries in the ODMG Type Systems" *Journal of Universal Computer Science*, volume 12, Issue No. 7, Pages 938-957, 2006.

[11] "Next-Generation Object Database Standardization" Date: 27-September-2007 *Object Database Technology Working Group White Paper. in International Multiconference Computer Science and Information Technology, IMCSIT 09*. ISBN**:** 978-1-4244-5314-6, 2009.

[12] K. Subieta, C. Beeri, F. Matthes, and J. W. Schmidt, "A Stack Based Approach to query languages," *in Proc. of 2nd Springer Workshops in Computing*, 1995.

[14] K. Subieta, "Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL)," *http://www.sbql.pl/overview/, 2008.*

[15] Bleja, M. Stencel, K. Subieta, K.,Fac. "Optimization of Object-Oriented Queries Addressing Large and Small Collections"

[16] Piotr Cybula and Kazimierz Subieta "Query Optimization Through Cached Queries for Object-Oriented Query Language SBQL" *Springer Journal Lecture Notes in Computer Science*, volume 5901/2010, pp.308-320, 2010.

[17] Piotr Cybula, Kazimierz Subieta "Decomposition of SBQL Queries for Optimal Result Caching" *Proceedings of the Federated Conference on Computer Science and Information Systems* pp. 841–848

[18] M. Tamer, Jose A. Blakeley "Query Processing in Object Oriented Database System" *in Proceeding of ACM SIGSOFT Software Engineering Notes* ,volume 35 , Issue No. 6, ISBN:0-201-59098-0, November 2010.

[19] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham, "Materialized view selection and maintenance using multi-query optimization," in Proc. of ACM SIGMOD, pp. 307–318, 2001.

[20] Belal Zaqaibeh and Essam Al Daoud, "The Constraints of Object-Oriented Databases**"** International Journal of Open Problems in Computer Science and Mathematics**,** *IJOPCM,* Volume 1, No. 1, June 2008