Query Optimization on Compressed and Decompressed Object-Oriented Database Using Operators

Abhijit Banubakode
Symbiosis International University (SIU)
Pune, India

Haridasa Acharya
SICSR, Pune
Pune, India

Abstract- **In this paper, we present an approach using various database operators that permits to enrich technique of query optimization existing in the object-oriented databases and the comparative analysis of query optimization of compressed and uncompressed object oriented database based on cost, cardinality and no of bytes. Focus is on query optimization using relational operator, logical operator and special operators. Our experimental study shows that the improvement in the quality of plans is significant only with decrease in cost, cardinality and no of bytes after database compression. Looking at the success of query optimization in the relational model, our approach inspires itself of these optimization techniques and enriched it so that they can support the new concepts introduced by the object oriented databases.**

Keywords: Query Optimization, Relational Databases, Object-Oriented Databases, Cost, Cardinality and Bytes

I. INTRODUCTION

Oracle allows arithmetic operators to be used while viewing records from a table or while performing data manipulation operation such as Insert, Update and Delete, addition, subtraction, division, Multiplication, Exponentiation and Enclosed operation. Oracle also uses Logical operator i.e. AND operator allows creating an SQL statement based on two or more conditions being met. It can be used in any valid SQL statement such as select, insert, update or delete. The AND operator requires that each condition must be meet for the record to be included in the result set. The OR condition allows creating an SQL statement where records are returned when any one of the condition are meet. The OR condition requires that any of the conditions must be meet for the record to be included in the result set. The oracle engine will process all rows in a table and display those records that do not satisfy the condition specified. In order to select data that is within a range of values, the BETWEEN operator is used. The BETWEEN operator allows the selection of rows that contain values within a specified lower and upper limit. NOT BETWEEN operator is used to select the values that are outside the range of values.

The comparison operator discussed above compared one value, exactly to one other value. Such precision may not always be desired or necessary for this LIKE predicate is use. The LIKE predicate allows comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters. The wildcard characters that are available are % and _ (underscore), % allows to match any string of any length (including zero length), _ (underscore) allows to match on a single character. The IN operator can be used to select rows that match one of the values included in the list in contrast NOT IN operator use to select only those rows that have a value not in the list. The query execution engine implements a set of physical operators. An operator takes execution engine as input one or more data streams and produces an output data stream. Examples of physical operators are (external) sort, sequential scan, index scan, nested loop join, and sort-merge join. We refer to such operators as physical operators since they are not necessarily tied one-to-one with relational operators. The simplest way to think of physical operators is pieces of code that are used as building blocks to make possible the execution of SQL queries. An abstract representation of such an execution is a physical operator tree, as illustrated in Fig.1. The edges in an operator tree represent the data flow among the physical operators. We use the terms physical operator tree and execution plan (or, simply plan) interchangeably. The responsible for the execution of the plan that results in generating answers to the query. Therefore, the capabilities of the query execution engine determine the structure of the operator trees that are feasible. We refer the reader to [7] for an overview of query evaluation techniques.

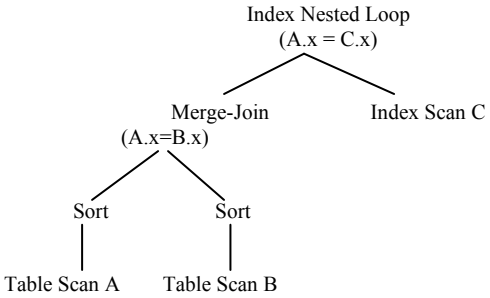

Fig 1: Operator Tree

The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of a SQL query as input and is responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans.
The task of an optimizer is nontrivial since for a given SQL query; there can be a large number of possible operator trees:

- . The algebraic representation of the given query can be transformed into many other logically equivalent algebraic representations: e.g.,
  Join (Join (A, B), C) =Join (Join (B, C), A)

- For a given algebraic representation, there may be many operator trees that implement the algebraic expression, e.g. typically there are several join algorithms supported in a database system.[6]

In this paper we consider compressed and decompressed object oriented database of retail banking system constructed various query plans for the same and presented query performance analysis**.** This paper is organized as follows. Next two sections describe Application and Preliminaries Notation. Section IV reviews the table compression technique. Section V establishes the Experimental setup. Section VI includes the table compression aspects. Section V discusses about query optimization process. Section VI presents analysis and result Finally, Section VIII concludes the paper**.**

II. APPLICATION

Fig1.shows an example of retail banking system. The bank is organized into various branches and each branch located in a particular city and monitors the assets. Bank customers are identified by their cust-id values. Bank offers two type of accounts i.e. saving account & checking account with loan facility thus the relation and attributes in the schema are [8]:

TRANS_DET (Trans_no, Inst_no, inst_dept, Inst_clr_dt,….)

FDSLAB_MAST (Fdslab_No, Minperiod, Maxperiod…….)

TRANS_MAST (Trans_no, Acct_Mo, Dt, Type, Dr_cr, Amt….)

FD_DET (Fd_ser_no, Fd_no, Type, Payto_acctno, Period…)

ACCT_MAST (Acct_no, Sf_no, Lf_no, Branch_no,……)

ACCT_FD_CUST_DET(Acct_fd_no, Cust_no)

FD_MAST (Fd_Ser_No, Sf_No, Branch_No, Intro_Cust_No…**)**

CUST_MAST(Cust_No, Fname, Mname, Lname, Dob_Inc…..**)**

NOMINEE_MAST (Noinee_No, Acct_Fd_No, Name……**)**

BRANCH_MAST (Branch_No, Name……**)**

SPRT_DOC (Acct_Code, Type, Docs……**)**

ADDR_DET (Addr_No, Code_No, Addr_Type, Area_1, …**)**

CNCT_DET(Addr_No, Code_No, Cntc_Type, Cntc_Data……..**)**

EMP_MAST(Emp_No, Branch_No, Fname, Mname, Dept…….**)**

Fig 2: Object considered for Banking System

III. PRELIMINARIES AND NOTATION

Operators are used in condition that compares one expression to another value or Expression [10]. They are used in the WHERE clause in the following format:

Syntax:
. . . . WHERE *expr operator value*

For Example
. . . . . WHERE hire_date = ’01-JAN-95’
. . . . . WHERE salary >= 6000
. . . . . WHERE last_name = ’Smith’

Fig 3: The typical query under consideration

IV. TABLE COMPRESSION

In today's environment, data warehouses of hundreds of terabytes have become increasingly common. To manage disk capacity, the table compression feature introduced in Oracle9i Release 2 can significantly reduce the amount of disk space used by database tables and improve query performance in some cases. The Oracle9i Release 2 table compression feature works by eliminating duplicate data values found in database tables. Compression works at the database block level. When a table is defined as compressed, the database reserves space in each database block to store single copies of data that appear in multiple places within that block. This reserved space is called the symbol table. Data tagged for compression is stored only in the symbol table and not in the database rows themselves. As data tagged for compression appears in a database row, the row stores a pointer to the relevant data in the symbol

table, instead of the data itself. The space savings come from eliminating redundant copies of data values in the table. The effects of table compression are transparent to a user or an application developer. We did table compression and obtained the different query plans which are discuss in next section. [9] To illustrate the regular RDBMS and the object oriented query optimizations, we consider schema of typical Retail Banking System**,** as the database

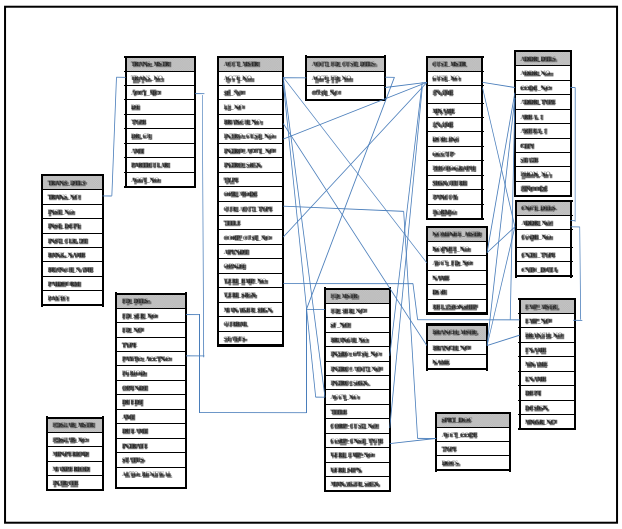

Fig 4: Object Oriented Schema for Banking System

V. OPTIMIZATION

A. Optimizer Hints

Hints make decisions usually made by the optimizer. As an application designer, user might know information about data that the optimizer does not know. Hints provide a mechanism to the optimizer to choose a certain query execution plan based on the specific criteria. [14]

Type of Hints

Hints falls into the following general classifications:

-Single-table

Single-table hints are specified on one table or view. INDEX and USE_NL are examples of single-table hints.

˜ Multi-table

Multi-table hints are like single-table hints, except that the hint can specify one or more tables or views. LEADING is an example of a multi-table hint. Note that USE_NL (table1 table2) is not considered a multi-table hint because it is actually a shortcut for USE_NL (table1) and USE_NL (table2).

˜ Query block

Query block hints operate on single query blocks. STAR_TRANSFORMATION and UNNEST are examples of query block hints.

-Statement

Statement hints apply to the entire SQL statement. ALL_ROWS is an example of a statement hint

Hint Syntax

User can send hints for a SQL statement to the optimizer by enclosing them in a comment within the statement. A block in a statement can have only one comment containing hints following the SELECT, UPDATE, MERGE, or DELETE keyword.

The following syntax shows hints contained in both styles of comments that Oracle supports within a statement block.

{DELETE|INSERT|MERGE|SELECT|UPDATE}/*+ hint [text] [hint [text]]... */ or

{DELETE|INSERT|MERGE|SELECT|UPDATE}--+ hint [text] [hint [text]]...

Where: ˜ DELETE, INSERT, SELECT, MERGE, and UPDATE are keywords that begin a statement block. Comments containing hints can appear only after these keywords.

˜ + causes Oracle to interpret the comment as a list of hints. The plus sign must immediately follow the comment delimiter; no space is permitted.

˜ hint is one of the hints discussed in this section. If the

The --+ hint format requires that the hint be on only one line.

The PARALLEL hint

The PARALLEL hints specify the desired number of concurrent servers that can be used for a parallel operation. The hint applies to the SELECT, INSERT, UPDATE, and DELETE portions of a statement, as well as to the table scan portion

TABLE1: QUERY PERFORMANCE COMPARISON OF COMPRESSED AND DECOMPRESSED OBJECT ORIENTED DATABASE

Query Performance on Decompressed Object Oriented Database					Query Performance on Compressed Object Oriented Database				
Query Analysis Using OR & LIKE Operator					Query Analysis Using OR & LIKE Operator				
Plans	**Indexing Hint**	**Cost**	**Card**	**Bytes**	**Plans**	**Indexing Hint**	**Cost**	**Card**	**Bytes**
Plan 1	-	**4**	**2**	**67**	**Plan 1**	-	**4**	**2**	**130**
Plan 2	**/*+ NO_CPU_COSTING */**	**4**	**2**	**67**	**Plan 2**	**/*+ NO_CPU_COSTING */**	**4**	**4**	**152**
Plan 3	--	**4**	**2**	**67**	**Plan 3**	--	**4**	**2**	**130**
Plan 4	**/*+ PARALLEL (CLIENT, 2) */**	**4**	**4**	**152**	**Plan 4**	**/*+ PARALLEL (CLIENT, 2) */**	**4**	**4**	**152**
Plan 5	**/*+ NO_CPU_COSTING */**	**4**	**2**	**67**	**Plan 5**	**/*+ NO_CPU_COSTING */**	**4**	**8**	**304**
Query Analysis Using BETWEEN &NOT BETWEEN Operator					Query Analysis Using BETWEEN &NOT BETWEEN Operator				
Plans	**Indexing Hint**	**Cost**	**Card**	**Bytes**	**Plans**	**Indexing Hint**	**Cost**	**Card**	**Bytes**
Plan 1	--	4	20	500	**Plan 1**	--	4	2	102
Plan 2	**/*+ PARALLEL (CLIENT, 2) */**	4	40	1000	**Plan 2**	**/*+ PARALLEL (CLIENT, 2) */**	4	28	700
Plan 3	**/*+ NO_CPU_COSTING */**	4	20	500	**Plan 3**	**/*+ NO_CPU_COSTING */**	4	14	250
Plan 4	--	4	12	300	**Plan 4**	--	4	2	102
Plan 5	**/*+ PARALLEL (CLIENT, 2) */**	4	24	600	**Plan 5**	**/*+ PARALLEL (CLIENT, 2) */**	4	28	700
Plan 6	**/*+ NO_CPU_COSTING */**	4	12	300	**Plan 6**	**/*+ NO_CPU_COSTING */**	4	14	250
Query Analysis Using IN &NOT IN Operator					Query Analysis Using IN &NOT IN Operator				
Plans	**Indexing Hint**	**Cost**	**Card**	**Bytes**	**Plans**	**Indexing Hint**	**Cost**	**Card**	**Bytes**
Plan 1	--	4	4	280	**Plan 1**	--	4	4	424
Plan 2	**/*+ PARALLEL (CLIENT, 2) */**	4	8	560	**Plan 2**	**/*+ PARALLEL (CLIENT, 2) */**	4	4	28
Plan 3	**/*+ NO_CPU_COSTING */**	4	4	280	**Plan 3**	**/*+ NO_CPU_COSTING */**	4	2	140
Plan 4	**/*+ PARALLEL (CLIENT, 2) */**	33	09	282	**Plan 4**	**/*+ PARALLEL (CLIENT, 2) */**	31	12	381
Plan 5	--	39	9	282	**Plan 5**	--	39	9	282

INDEX_FFS

The INDEX_FFS hint causes a fast full index scan to be performed rather than a full table scan.

INDEX

The INDEX hint explicitly chooses an index scan for the specified table. You can use the INDEX hint for domain, B-tree, bitmap, and bitmap join indexes. However, Oracle recommends using INDEX_COMBINE rather than INDEX for the combination of multiple indexes, because it is a more versatile hint.

NO_CPU_COSTING hint

This hint turns CPU costing off for the SQL statement. The optimizer uses the I/O cost model which measures everything in single block reads and ignores CPU cost.

INDEX_DESC

The INDEX_DESC hint explicitly chooses an index scan for the specified table. If the statement uses an index range scan, then Oracle scans the index entries in descending order of their indexed values

FIRST_ROWS (n)

The hints FIRST_ROWS (n) (where n is any positive integer) or FIRST_ROWS instruct Oracle to optimize an individual SQL statement for fast response. FIRST_ROWS (n) affords greater precision, because it instructs Oracle to choose the plan that returns the first n rows most efficiently. The FIRST_ROWS hint, which optimizes for the best plan to return the first single row, is retained for backward compatibility and plan stability.

USE_CONCAT

The USE_CONCAT hint forces combined OR conditions in the WHERE clause of a query to be transformed into a compound query using the UNION ALL set operator. Generally, this transformation occurs only if the cost of the query using the concatenations is cheaper than the cost without them.

VI. ANALYSIS: RESULT COMPARISON

We did an experimental study and achieved statistically significant improvement in the quality of some of query plans with a modest decrease in the optimization Cardinality and no Bytes. The experiments were conducted using oracle10g Table: 1 shows the Query Comparisons of decompressed and compressed Object Oriented Database Management System Based on Cost, Cardinality & No of Bytes using various operators. From experimental setup we observed that there is significant improvement after query optimization in object oriented database. Fig.5 shows Query performance histogram for OR & LIKE operator (Decompress OODB) and Fig.6 shows Query performance histogram for OR & LIKE operator (Compressed OODB). A histogram divides the values on a column into k buckets. In many cases, k is a constant and determines the degree of accuracy of the histogram. However, k also determines the memory usage, since while optimizing a query; relevant columns of the histogram are loaded in memory. There are several choices for "bucketization" of values. In many database systems, equi-depth (also called equi-height) histograms are used to represent the data distribution on a column. If the table has n records and the histogram has k buckets, then an equi-depth histogram divides the set of values on that column into k ranges such that each range has the same *number* of values, i.e., n/k. compressed histograms place frequently occurring values in singleton buckets. The number of such singleton buckets may be tuned. It has been shown in [13] that such histograms are effective for either high or low skew data. One aspect of histograms relevant to optimization is the assumption made about values within a bucket. For example, in an equi-depth histogram, values within the endpoints of a bucket may be assumed to occur with uniform spread. A discussion of the above assumption as well as a broad taxonomy of histograms and ramifications of the histogram structures on accuracy appears in [11]. In the absence of histograms, information such as the *min* and *max* of the values in a column may be used. However, in practice, the second lowest and the second highest values are used since the *min* and *max* have a high probability of being outlying values. Histogram information is complemented by information on parameters such as number of distinct values on that column although histograms provide information on a single column; they do not provide information on the *correlations* among columns. In order to capture correlations, we need the *joint* distribution of values. One option is to consider 2-dimensional histograms [12,13]. Unfortunately, the space of possibilities is quite large. In many systems, instead of providing detailed joint distribution, only summary information such as the number of distinct pairs of values is used. For example, the statistical information associated with a multi-column index may consist of a histogram on the leading column and the total count of distinct combinations of column values present in the data. If we compare between fig 5 and fig 6 we observed that cost and cardinality is same but the no of bytes required for query execution is doubled in compressed object oriented database. Plan 2 uses /*+ NO_CPU_COSTING */ hint. This hint turns CPU costing off for the SQL statement. The optimizer uses the I/O cost model which measures everything in single block reads and ignores CPU cost. In plan2 cost is same but cardinality and no of bytes is approximately doubled. In Plan 3 cost and cardinality is same but no of bytes required for query execution is doubled. Plan 4 uses /*+ PARALLEL (CLIENT, 2) */ hint. The PARALLEL hint specify the desired number of concurrent servers that can be used for a parallel operation. The hint applies to the SELECT, INSERT, UPDATE, and DELETE portions of a statement, as well as to the table scan portion. Plan 5 uses /*+ NO_CPU_COSTING */ hint here the cost is same but the cardinality and no of bytes are approximately doubled. Fig 7 shows Query performance histogram for BETWEEN & NOT BETWEEN Operator (Decompressed OODB) and Fig 8 shows Query performance histogram for BETWEEN & NOT BETWEEN Operator (Compressed OODB).If we compare between two histogram we observed that in Plan 1 cost is same but there is tremendous change in cardinality and no of bytes required for query execution as compare to compressed object oriented database. Plan 2 uses /*+ PARALLEL (CLIENT, 2) */ hint here cost is same but there is change in Cardinality and no of bytes required more for query execution .Plan 3 uses /*+ NO_CPU_COSTING */ hint here the cost is same but cardinality and no of bytes increased .In Plan 4 cost is same but there is change in Cardinality and no of bytes required more for query execution .Plan 5 uses /*+ PARALLEL (CLIENT, 2) */ hint. In plan5 cost is same but there is little change in cardinality and No of Bytes .Plan 6 uses /*+ NO_CPU_COSTING */ hint here the cost is same but the cardinality and no of bytes are also approximately equal. Fig 9 shows Query performance histogram Using IN & NOT IN Operator (Decompressed OODB) and Fig 10 shows Query performance histogram Using IN & NOT IN Operator (Compressed OODB).If we compare between Fig 9 and Fig 10 we observed that in Plan 1 cost and Cardinality is same but there is change in no of bytes required for query execution as compare to compressed object oriented database. Plan 2 uses /*+ PARALLEL (CLIENT, 2) */ hint. Here cost is same but Cardinality is doubled and there is tremendous change in no of bytes required more for query execution .Plan 3 uses /*+ NO_CPU_COSTING */ hint. Here the cost is same but cardinality and no of bytes required for query execution is doubled. Plan 4 uses /*+ PARALLEL (CLIENT, 2) */ hint. In plan4 cost is slightly change cardinality is decreased and No of Bytes increased as compare to compressed object oriented database. In Plan 5 cost, cardinality and no of Bytes required for execution are same. From the result we could conclude that after object oriented database compression cost is not affected but cardinality and no of bytes are affected but it is not with all the cases in some of queries there is no major change in all three attributes.

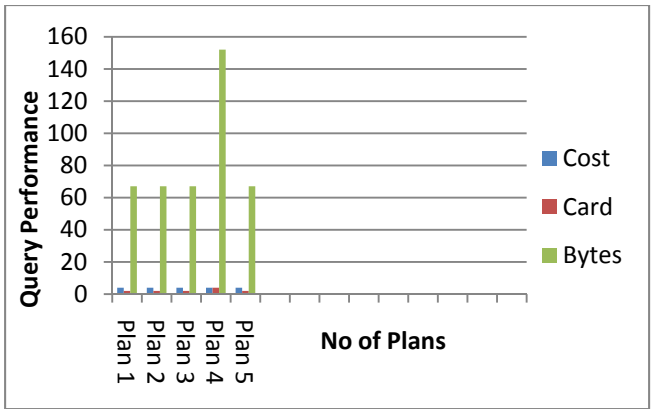

Fig. 5: Query performance histogram for OR & LIKE operator (Decompress OODB)

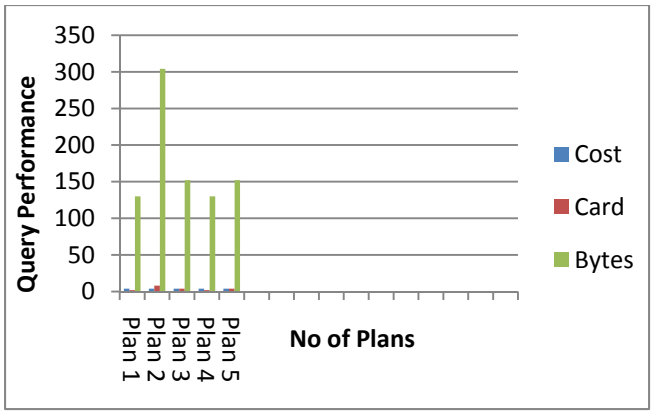

Fig. 6: Query performance histogram for OR & LIKE operator (Compressed OODB)

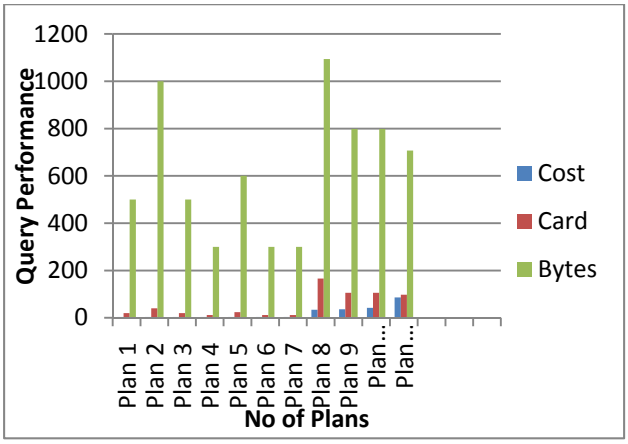

Fig. 7: Query performance histogram for BETWEEN & NOT BETWEEN Operator (Decompressed OODB)

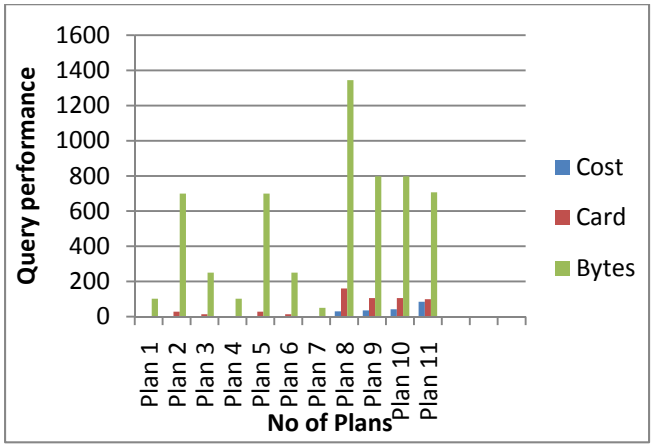

Fig. 8: Query performance histogram for BETWEEN & NOT BETWEEN Operator (Compressed OODB)

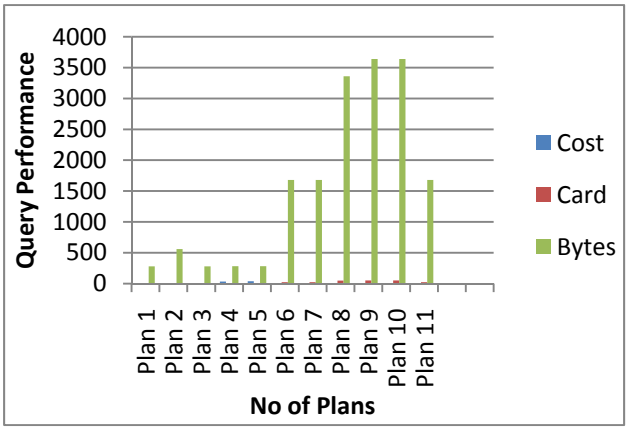

Fig. 9: Query performance histogram Using IN & NOT IN Operator (Decompressed OODB)

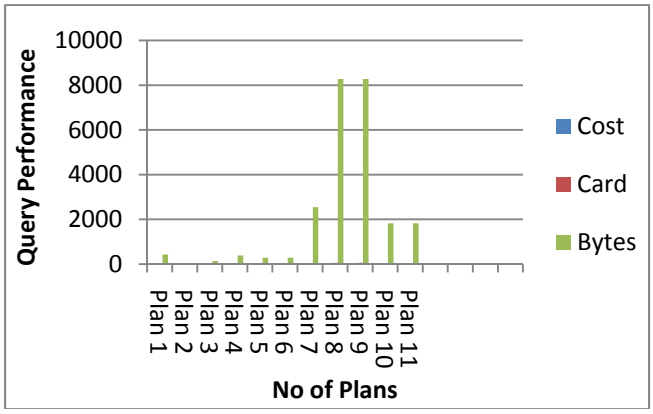

Fig. 10: Query performance histogram Using IN & NOT IN Operator (Compressed OODB)

VII. FUTURE WORKS

In this paper we have considered very few operators. Our work is so far has opened up a number of interesting future research directions such as Query Optimization on Compressed and Decompress Object-Oriented Database Using Union, Intersect, Minus and All operators that we are planning to pursue in future.

VIII. CONCLUSIONS

One of the biggest problems in Object Oriented Database is the optimization of queries. Due to these problems optimization of object-oriented queries is extremely hard to solve and is still in the research stage. This work is expected to be a significant contribution to the object database management area which will not only reduce time or efforts but will also improve the quality. From above results we could conclude that first in object oriented database management system there is no significant cost reduction after query optimization using operators and second cardinality and no of bytes in the object oriented database does affected after query optimization when the indexing methods are change.

REFERENCES

[1] PL/SQL User's Guide and Reference 10*g* Release 1 (10.1) Part Number B10807-01
[2] Oracle ® Database Performance Tuning Guide10*g* Release 1 (10.1) Part No. B10752-01 December 2003
[3] Abhijit Banubakode and Haridasa Acharya, "Query Optimization in the Object Oriented Database Using Nested Query", Proc. 3rd International Conference on Computer Modeling and Simulation ICCMS 2011 January 7 - 9, 2011, Mumbai, India
[4] Abhijit Banubakode and Haridasa Acharya, "Query Optimization in the Object-Oriented Database Using Views", Proc. International Conference On Computing ICC 2010,New Delhi 27-28 December 2010
[5] Abhijit Banubakode and Seema Kedar "Query Optimization in Compressed Database System" International Conference on Advance Computing (ICAC- 2008)" ACM Students Chapter Department of Computer Science and Engineering Anuradha Engineering College Chikhli-443 201, Maharashtra, India
[6] Surajit Chaudhuri, "An Overview of Query Optimization in Relational Systems" Microsoft Research One Microsoft Way Redmond, WA 98052
[7] Graefe G. "Query Evaluation Techniques for Large Databases" In ACM Computing Surveys: Vol 25, No 2. June 1993.
[8] Ivan Bayross,SQL,PL/SQL The programming language of oracle,BPB Publication
[9] http://shaharear.blogspot.com/2008/10/table- compression.html
[10] Oracle ® Database Performance Tuning Guide10*g* Release 1 (10.1) Part No. B10752-01 December 2003
[11] Poosala, V., Ioannidis, Y.E., Haas, P.J., Shekita, E.J. Improved Histograms for Selectivity Estimation of Range Predicates In Proc. of ACM SIGMOD, Montreal, 1996.
[12] Muralikrishna M., Dewitt D.J. Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, Proc. of ACM SIGMOD, Chicago, 1988.
[13] Poosala, V., Ioannidis, Y.E. Selectivity Estimation without the Attribute Value Independence Assumption. In Proc. of VLDB, Athens, 1997.
[14] "Understanding Optimizer hints", Oracle database Performance Tuning Guide

Abhijit Banubakode received ME degree in Computer Engineering from Pune Institute of Computer Technology (PICT), University of Pune, India in 2005 and BE degree in Computer Science and Engineering from Amravati University, India, in 1997. Presently he is perusing his Ph.D. from Symbiosis Institute of Research and Innovation (SIRI), a constituent of Symbiosis International University (SIU), Pune, India. His current research area is Query Optimization in Compressed Object-Oriented Database Management Systems (OODBMS). Currently he is working as Assistant Professor in Department of Information Technology, Rajarshi Shahu College of Engineering, Pune, India .He is having 13 years of teaching experience. He is a member of International Association of Computer Science and Information Technology (IACSIT), ISTE, CSI and presented six papers in International and National conference.

Dr. Haridasa Acharya received MSc degree in Applied Mathematics from University of PUNE, India in the year 1970 and the PhD degree in Mathematics from Indian Institute of Technology (IIT), Kanpur, India in 1975. He is an Associate Professor at Symbiosis Institute of Computer Studies and Research, a constituent of Symbiosis International University (SIU), Pune, India. Was a National Fellow of Biotechnology (Dept of Science and Tech.) in the year 1990 at IASRI, New Delhi. He has worked as principal investigator in research projects funded by ICAR, UGC. He worked as a co-investigator in many ICAR research projects and the advisor for Design of Experiments and Research Analysis. Had opportunity to guide research scholars working in diversified areas like Food Technology, Veterinary Medicine and Sciences, Soil Science and Farm Engineering apart from students in Computer Science and Mathematics. His current area of research is fuzzy protocols, query optimization, and analytics. He was the head of the Dept. of Basic Sciences and Computers at College of Agric. Engg., under the Marathwada Agric. University for period of twenty years, and has been a faculty and Program Head (MSc) at SICSR for the past three years. He has more than 30 scientific research papers, in national and international journals to his credit; in addition he presented papers at National and International conferences. He was awarded Shiksha Ratna by India International Friendship Society, in Nov 2008, for his significant contribution to Education.