Query Optimization in Object-Oriented Database Management Systems: A short review

Prof.Abhijit Banubakode
Symbiosis International University(SIU)
Pune, India

Dr. Haridasa Acharya
Allana Institute of Management & Science
Pune, India

***Abstract*-Object-oriented database systems began developing in the mid-80's out of a necessity to meet the requirements of applications beyond the data processing applications which were [are] served by relational database systems. We propose in this paper a new approach that permits to enrich technique of query optimization existing in the object-oriented databases and the comparative analysis of query optimization for relational databases and object oriented database based on cost, cardinality and no of bytes. Seen 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 databases.**

Keywords: Query Optimization, Relational Databases,

Object-Oriented Databases

I. INTRODUCTION

Query optimization is the process of selecting the most efficient query-evaluation plan from many strategies usually possible for processing a given query if the query is complex. One aspect of optimization occurs at the relational-algebra level, where the system attempts to-find an expression that is equivalent to given application, but more efficient to execute. Another aspect is selecting a detailed strategy for processing the query, such as choosing the algorithm to use for executing the operation, choosing the specific indices to use, and so on. In either case the problem boils down to parsing, estimating complexity of the algorithms which minimize cost, or time as the case may be. Object-oriented databases integrate object orientation with database capabilities. Object orientation allows a more direct representation and modeling of real-world problems. Today Oracle, Microsoft, Borland, Informix, and others incorporated object-oriented features into their relational systems. Most current OODBs are still not full-fledged database systems comparable to current relational database systems (RDBs) [8].

II. BASIC CONCEPTS

The Object-Oriented Database Model
The OODB model is based on a number of basic concepts, namely Object, Class, Abstraction, Encapsulation, Inheritance and Polymorphism. [8]

Object: An object is an entity that has a well defined state and behavior associated with it. The state of an object includes the current values of all its attributes. Behavior is how an object acts or reacts, in terms of its state changes and operations performed upon it. Each object has unique Object identifier (OID) which is automatically generated by the system. So that the objects can be easily identified. This is similar to a primary key in the relational model.
Class: 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.
Abstraction: Abstraction is the process of identifying the key aspects of an entity and ignoring the rest.
Encapsulation: Encapsulation means hiding the data members of an object from an object from the outside world.
Inheritance: Inheritance is a property of a class hierarchy whereby each subclass inherits attributes and methods of its super-class
Polymorphism: The ability of different objects to respond to the same message in different ways is called polymorphism.

III. QUERY OPTIMIZER COMPONENT

The Query optimizer consist of three major components [8]

A SQL Transformation

B Execution Plan Selection

C Cost Model and Statistics

A SQL Transformation
The purpose of SQL Transformation is to transform the original SQL statement into semantically equivalent SQL statement that can be processed more efficiently.

B Execution Plan Selection
In Execution Plan Selection, the optimizer selects an execution plan. That describe all the steps when the SQL is processed, such as order in which table are accessed, how the table are join together.

C Cost Model and Statistics
The Cost Estimates Are Base Upon I/O, CPU And Memory Resources Required By Each Query Operation, And The Statistical Information About The Database Object Such As Table, Indexes And Views.

IV. OPTIMIZATION PROCESS

The query optimization is the processes of selection of the best path of access data in a database. [4]Process of optimization is summarizes in three steps (See fig 1) Rewrite step consist in a syntactic and semantic rewrite of the query in the goal to determine simpler equivalent queries. The result of this step is the generation of a query graph. Ordering operation step is takes place in two phases: generation and assessment of plans which determined in the first phase. Execution step permits to choose the optimal execution plan and to execute it]

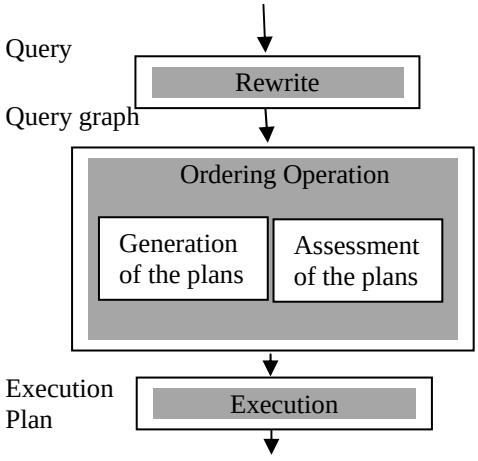


Fig1: Optimization Process

Problems of Object Model

Minimal query optimization: One of the biggest problems in OODBs is the optimization of queries. The additional complexity of the object-oriented data model (OODM) complicates the optimization of OODBs queries. [4]

This additional complexity is due to:

- Additional data types
- Complex objects
- Methods and Encapsulation

ODBs query languages support the use of nested structures, which may again highly complicate the optimization process. Due to these problems optimization of object-oriented queries is extremely hard to solve and is still in the research stage. The optimization of joins is also another issue that needs more attention.

Lack of query facilities: The OODB query language is not ANSI sql compatible. The query language do not support Nested sub-queries, Set queries like Union, Intersection, Difference

Security concerns with OODBs: RDBs support authorization, OODBs do not support authorization. RDBs allow users to grant and revoke privileges to read or change the definitions, this feature has to be improved by OODB.

No support for dynamic class definition changes with OODBs: Most OODBs do not allow dynamic changes to the database schema, such as Adding a new attribute or method to a class,

- Adding a new superclass to a class,
- Dropping a superclass from a class,

A Explain Plan

The Explain Plan is the sequence of operations performed by oracle to execute the statement. By examining the explain plan, we can identify inefficient SQL statements [12]

The explain plan gives the following information: An ordering of the tables referenced by the statement An access method for each table mentioned in the statement Data operations like filter, sort, or aggregation Optimization, such as the cost and cardinality of each operation

In order to get the result of EXPLAIN PLAN execute the following query

SELECT id, object name, operation, options FROM PLAN_TABLE order by id

The table shows the result of explain plan:--

ID	OBJECT_NAME	OPTIONS	OPERATION
0		SELECT STATEMENTS	
0		SELECT STATEMENT	
0		SELECT STATEMENT	
1		SORT	GROUP BY
1		SORT	GROUP BY
2	RBRANCH	BY INDEX ROWID	TABLE ACCESS
2	RBRANCH	BY INDEX ROWID	TABLE ACCESS

Fig 2: Explain plan output

The fig consist of:

ID: is the number assigned to each step in the execution plan.

OBJECT_NAME: is the name of table or index,

OPTIONS: Options tell more about an operation. For example, the operation TABLE ACCESS can have the options: FULL or BY ROWID. Full means, the entire table is accessed whereas BY ROWID means, Oracle knows from which block the rows are to be retrieved, which makes the time to access the table shorter.

OPERATION: Provides methods for retrieving and processing rows from a table.

B Application

We are considering an example of retail banking system. The bank is organized into various branches and branch 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:

Customer (cust_name, cust_street, cust_city)

Branch (branch_city, branch_name, assets)

Account (acct_no, branch-name, and balance)

Depositor (cust_name, acct_no)

Loan (loan_no, branch_name, amount)

Borrower (cust_name, loan_no)

Fig 3: Banking system considered

C Transformations

We make the key observation that since a group-by reduces the cardinality of a relation; an early evaluation of group-by could result in potential saving in the costs of the subsequent joins. We present an example that illustrates a transformation based on the above observation. An appropriate application of such a transformation could result in plans that are superior to the plans produced by conventional optimizers by an order of magnitude or more.

Example2.3: Let us consider the query that computes branches located in a particular city and total count of branches in each city .The following alternative plan is possible. First, group-by clause applied after condition and hence search time is more and CPU cost is high. In other words we first check the condition and then grouped on branch city. Second, group-by clause applied before condition hence search time is less and CPU cost is less. Here we grouped on branch city first and then check the condition

D Related Work

In a recent paper [16], Yan and Larson identified a transformation that enables pushing the group-by past joins. Their approach is based on deriving two queries, one with and the other without a group-by clause, from the given SQL query. The result of the given query is obtained by joining the two queries so formed. Thus, in their approach, given a query, there is a unique alternate placement for the group-by operator. Observe that the transformation reduces the space of choices for join ordering since the ordering is considered only within each query. Prior work on group-by has addressed the problem of pipelining group-by and aggregation with join [5, 6] as well use of group-by to flatten nested SQL queries [7, 5, 8, and 9]. But, these problems are orthogonal to the problem of optimizing queries containing group-by clause.

E Preliminaries and Notation

We will follow the operational semantics associated with SQL queries [10, 11]. We assume that the query is a single block SQL query, as below.

Select All <columnlist> AGG1 (bl) AGG2 (bn)
From <tablelist>
Where cond1 And cond2 . . . And condn
Group By col1,..col2

Fig 5: The typical query under consideration

The WHERE clause of the query is a conjunction of simple predicates. SQL semantics require that <columnlist> must be

among col1,.. colj. In the above notation,AGGi…..AGGn represent built-in SQL aggregate functions. In this paper, we will not be discussing the cases where there is an ORDER BY clause in the query. We will also assume that there are no nulls in the database. These extensions are addressed in [12]. We refer to columns in {b1, ..bn} as the aggregating columns of the query. The columns in (col1, ..colj} are called grouping columns of the query. The functions{AGG1, ..AGGn} are called the aggregating functions of the query. For the purposes of this paper, we included Avg and Count as well as cases where the aggregate functions apply on columns with the qualifier

Optimization To illustrate the object oriented query optimizations consider the same example of Retail Banking system.

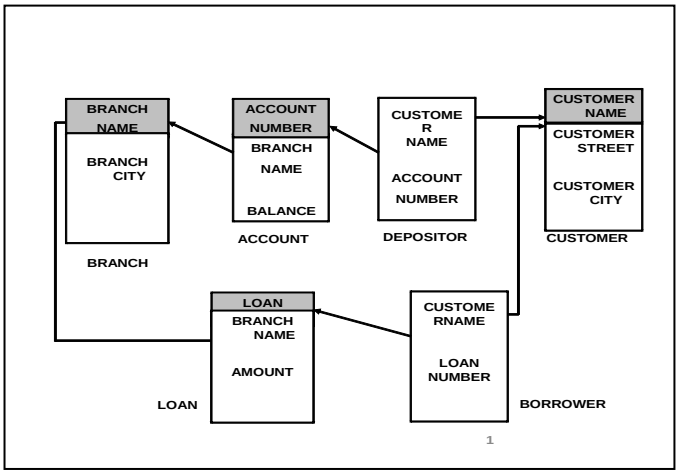


Fig 4: Retail Banking Schema

A. Query Optimization in OODB

As an example, consider the same Query suppose we want to find the number of branch in each city except pune

Creation of Object Oriented Type

```
Create Type Branchdet_Ty as Object
(Branch_City Varchar2 (30),
Assets  Number (26, 2));

Create Type Accountdet_Ty as
Object (Branch_Name  Varchar (30),
 Balance  Number (12, 2));
 Creation of Object Oriented Table

Create Table Branch1 (Branch_Name  Varchar2 (30)
Primary Key,
 Branchdetail  Branchdet_Ty);

Create Table  Account1
(Account_Number  Varchar(15),
Accountdetail  Accountdet_Ty);
```

As an example, consider the above Query suppose we have to find the number of branch in each city except pune

The query evaluation plans for OODB are: -

```
∏branchdetail.branch_city, count (*)
|
σ Branchdetail.branch-city! =pune
|
Branch1
|
group-by branchdetail. branch-city
```

Plan1

```
∏ branchdetail.branch_city, count (*)
|
group-by group-by branch-city
|
σ branchdetail.branch-city!=pune
|
Branch1
```

Plan 2

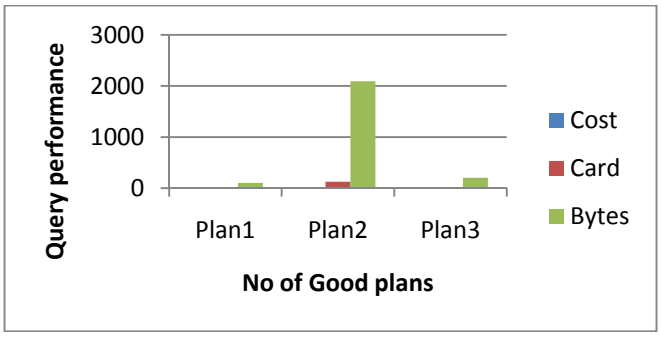
```
∏ branchdetail.branch_city, count (*)
|
σ branchdetail.branch-city<‘pune’ or
|   Branchdetail.branch-city > ‘pune’
|
branch1
|
group-by branchdetail.branch-city
```

Plan 3

V. EXPERIMENTAL SETUP

We did an experimental study. We achieved statistically significant improvement in the quality of plans with a modest decrease in the optimization cost. The experiments were conducted using on oracle database Table: 1 shows the Query performance of OODBMS Based on Cost, Cardinality & No of Bytes. From experimental setup we observed that there is significant improvement after query optimization in object oriented database.

Table 1: Query Performance of OODB for GROUP BY Clause



Object Oriented Database (OODB)			
Group By Clause			
Plans	Query Performance		
	Cost	Card	Bytes
Plan1	11	8	102
Plan2	11	123	2091
Plan3	11	12	204

Fig 6: Query performance Histogram for OODB

VI. COST ESTIMATION

Given a query there are many equivalent alternative algebraic expression for each expression there are many ways to implement them as operators. The cost estimates are based upon I/O, CPU and Memory resources required by each query operation, and the statistical information about the database object such as Table, Indexes and Views. In a large number of systems, information on the data distribution on a column is provided by *histograms*. Fig 3 shows a histogram for query performance for 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 [13]. 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 [15, 16]. 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.

VII. 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 Database Management area which will not only reduce time or efforts but will also improve the quality and will reduce the cost.

REFERENCES

[1] Oracle Advance Compression, an Oracle White paper, April 2008.
[2] Surajit Chaudhuri," An Overview of Query Optimization in Relational Systems", Microsoft Research One Microsoft Way Redmond WA 98052, +1-(425)-703-2001

[3] Yannis E. Ioannidis "Query Optimization", Computer Science Department, University of Wisconsin, Madison, WI 53706
[4] Minyar Sassi, and Amel Grissa-Touzi "Contribution to the Query Optimization in the Object-oriented Databases" Proceedings of world academy of Science, Engineering and Technology Volume 6 ISSN June 2005 1307-6884
[5] Yannis E. Ioannidis "Query Optimization", Computer Science Department, University of Wisconsin, Madison, WI 53706
[6] http://elearning.tvm.tcs.co.in/dbms/62.htm
[7] Journal of Object Technology - Achievements and Weaknesses of Object-Oriented.htm BY Sikha Bagui, Department of Computer Science, University of West Florida, U.S.A.
[8] Query Optimization in Oracle Database 10g Doc**umentation Library**
[9] SYBASE, Performance and Tuning:Optimizer and Abstract Plans Adaptive Server® Enterprise12.5.1
[10] Oracle Advanced Compression An Oracle White Paper April 2008
[11] "Oracle Query Optimization Tools" ECS 165A Database Systems, Winter 2004
[12] Surajit Chaudhuri," An Overview of Query Optimization in Relational Systems", In Proc.of ACM SIGMOD,San Francisco,1987
[13] Tsang A., Olschanowsky M."A Study of Database 2 Customer Queries," IBM Santa Teresa Laboratory, TR-o3.413.
[14] Chaudhuri, S., Shim K."Including Group-By in Query Optimization." In Proc. Of VLDB, Santiago, 1994.
[15] Yan W. P., Larson P. A., "Performing Group-By before Join," International Conference on Data Engineering, Feb. 1993, Houston.
[16] DayaI U. "Of Nests and Trees: A Unified Approach to Processing Queries that contain sub queries, aggregates and quantifiers," in Proceedings of the 13th VLDB, Aug 1987.
[17] Klug A. "Access Paths in the ABE Statistical Query Facility," in Proceedings of 1982 ACMSIGMOD Conference on the Management of Data.
[18] Kim W. "On Optimizing an SQL-like Nested Query," in ACM Transactions on Database Systems, 7(3):443-469, Sep 1982.
[19] Ganske R. A., Wong H. "Optimization of Nested Queries Revisited," in Proceedings of 1987 ACMSIGMOD Conference on Management of Data, San Francisco, May 1987.
[20] Murahkrishna M. "Improved Unnesting Algorithms for Join Aggregate SQL Queries," in Proceedings of the 18th VLDB, 1992
[21] Date C. J., Darwen H. "A Guide to the SQL Standard: A User's Guide," Addison-Wesley, 1993.
[22] ISO. Database Language SQL ISO/IEC, Document ISO/IEC 9075:1992. Also available as ANSI Document ANSI X3.135-1992.
[23] Chaudhuri S., Shim K. "The promise of Early Aggregation," HPL Technical Report, 1994.
[24] 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.
[25] Muralikrishna M., Dewitt D.J. Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, Proc. of ACM SIGMOD, Chicago, 1988.
[26] Muralikrishna M., Dewitt D.J. Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, Proc. of ACM SIGMOD, Chicago, 1988.
[27] Poosala, V., Ioannidis, Y.E. Selectivity Estimation without the Attribute Value Independence Assumption. In Proc. of VLDB, Athens, 1997.

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. e 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.

Abstract In many relational database systems using a relational algebra-based query language, query optimization involves the syntactic modication of queries into a â€œcanonical formâ€[Dat90], and then choosing from possibly several methods of evaluating the query. Semantic query optimiza-tion (SQO) is the idea of semantically transforming a query using additional schema informa-tion, such as integrity constraints. The query is passed through three dierent phases of logical transformation: standardization, simplication, and amelioration.Â 1 Motivation for Query Optimization. 95. 2 Literature Review. 97. 2.1 A Taxonomy of Query Transformations . . . In this paper, query processing and optimization in object-oriented database systems (OODBs) in a centralized environment is discussed. The typical chain query processing and optimization in OODBs is...Â Selinger, P. G. and M. Adiba, â€œAccess Path Selection in Distributed Database Management Systemsâ€, IBM Research Lab., San Jose, Cal., RJ2883(36439), August 1980Google Scholar. STON86. Stonebraker, M. and L. Rowe, â€œThe Design of POSTGRESâ€,ACM SIGMOD86,Washington, D.C., May 1986, pp 340â€“355Google Scholar. Historically, database system imple-mentations and research have focused on the row-by row data layout, since it performs best on the most common application for database systems: business transactional data processing. However, there are a set of emerging applications for database systems for which the row-by-row layout performs poorly. These applications are more analytical in nature, whose goal is to read through the data to gain new insight and use it to drive decision making and planning. In this dissertation, we study the problem of poor performance of row-by-row data layout for these em...