A Study on the Role of Equivalence Rules in the Enhancement of Query Performance

Tejy Johnson¹, Dr.S.K.Srivatsa²

¹Research Scholar, Department of Computer Applications, Dr.MGR University, Maduravoyal, Chennai, India.
²Senior Professor, Department of Computer Applications St.Joseph College of Engineering, Chennai-119, India.

ABSTRACT

For the past decade many research and development work had been carried out in the area of Query processing. As Query Processing is an important concept in database technology. In Query Processing one of the most critical and important step is Query Optimization. Many Query Optimization algorithms, techniques and methods have been evolved to optimize a query efficiently. In spite of all these techniques and methods, optimizing a query accurately was not possible. The main reason behind this was, we required sufficient information to determine which technique would be most effective. To determine this was a difficult task. This problem can be solved by understanding the fact that one aspect of optimization occurs at the relational algebra level where the system attempts to find an expression that is equivalent to the given expression, but more efficient to execute. To transform a given expression into an equivalent one we can make use of Equivalence Rules. We can use these rules to generate systematically all expressions equivalent to the given query. This paper focuses on new directions in the area of Query Optimization. Here we have discussed about the vital role of Equivalence Rules in optimizing a query efficiently. In this paper we have also briefly discussed about the cost estimation which is one of the factors that influences query performance. Here we have also given some best practices that have to be considered in reducing the Execution Cost of a query.

Keywords: Query Optimization, Query Processing, Query Execution plans, Equivalence rules and Execution cost

1.INTRODUCTION

Query Processing and Optimization process work together to execute any kind of queries. Query Processing is concerned with execution of query or refers to the activities involved in extracting data from a data warehouse [1, 2]. It is a process of translating a query expressed in a high-level language such as SQL into low-level data manipulation operations. Typically Query Processing involves many steps [3, 4]. The first step is query decomposition in which an SQL query is first scanned, parsed and validated. The scanner identifies the language tokens such as SQL keywords, attribute names and relation names in the text of the query, whereas the parser checks the query syntax to determine whether it is formulated according to the syntax rules of the query language. The query must also be validated by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried. An internal representation of the query is then created. A query expressed in relational-algebra is usually called initial algebraic query and can be represented as a tree data structure called query tree. It represents the input relations of the query as leaf nodes of the tree and represents the relational-algebra operations as internal nodes. The quality of an algebraic query is defined in terms of expected performance. Therefore the second step is Query Optimization step that transforms the initial algebraic query using relational-algebra transformations into other algebraic queries until the best one is found. A Query Execution Plan is then founded. It is represented as a query tree which includes information about the access method available for each relation as well as the algorithms used in computing the relational operations in the tree. The next step is to generate the code for the selected Query Execution Plan. This code is then executed in either compiled or interpreted mode to produce the query result.

Query Optimization plays a vital role in Query Processing [5, 6]. Query Optimization can be formally defined as a process of transforming a query into an equivalent form which can be evaluated more efficiently. The goal of Query Optimization is to reduce the system resources required to fulfill a query and ultimately provide the user with the correct result set faster.[7] Query Optimization is of great importance in database management system for the following reasons. First it provides the user with faster results. Secondly it allows the system to service more queries in the same amount of time. Thirdly it allows the server to run more efficiently. The Query Optimization process consists of getting a query on n relations and generating the best Query Execution Plan.
The structure of the paper is as follows. Section 2 describes generation of equivalent expression. In section 3 we discuss the role of Equivalence Rules in the enhancement of Query Optimization. Section 4 discusses about the cost estimation. In section 5 we have presented some best practices to be followed to reduce the Execution Cost. Finally we have concluded our work in section 6.

2. Generation of Equivalent Expression

As we mentioned one aspect of optimization occurs at relational-algebra level where the system transforms initial expression into an equivalent expression which is more efficient to execute.[8] Two relational-algebra expressions are said to be equivalent if the two expressions generate two relation of the same set of attributes and contain the same set of tuples although their attributes may be ordered differently.[9,10] The query tree is a data structure that represents the relational-algebra expression in the Query Optimization process. The leaf nodes in the query tree correspond to the input relations of the query. When executing the query the system will execute an internal node operation whenever its operands are available then the internal node is replaced by the relation that is obtained from the preceding execution.

Consider the relational algebra expression for the query “find the names of all customers who have an account at any branch located in Brooklyn”.

\[ \Pi \text{Customer-name} \left( \sigma \text{city} = \text{"Brooklyn"} \right) \left( \text{branch account Depositor} \right) \].

This expression constructs a large intermediate relation branch account depositor. By reducing the number of tuples of the branch relation that we need to access, we reduce the size of the intermediate result. Then the query is now represented by the relational algebra expression as

\[ \Pi \text{Customer-name} \left( \sigma \text{city} = \text{"Brooklyn"} \right) \left( \text{branch account Depositor} \right) \].

This is equivalent to our original algebra expression, but which generates smaller intermediate relations. This can be depicted by the diagram which is as follows.

![Figure 1](image)

Figure 1. (a) Initial expression  
Figure 1. (b) Transformed expression

Given a relational-algebra expression, it is the job of the optimizer to come up with a Query Evaluation Plan that computes the same result as the given expression and find the least costly way of generating the result[11,12]. To find the least costly Query Evaluation Plan the optimizer needs to generate alternative plans that produce the same result as the given expression and choose the least costly one.

Generation of Query Evaluation Plans involves three steps: (1) Generating expressions that are logically equivalent to the given expression, (2) Estimating the cost of each evaluation plan, and (3) Annotating the resultant expressions in alternative ways to generate alternative Query Evaluation Plans.

To implement the first step, the query optimizer must generate expressions equivalent to a given expression. It is done by means of Equivalence Rules that specify how to transform an expression into a logically equivalent one.

3. Role of Equivalence Rules

An [13] Equivalence Rules say that if the expression of two forms are equivalent then we can replace an expression of the first form by an expression of the second form or vice versa. The optimizer uses Equivalence Rules to transform expressions into other logically equivalent expressions. There are around 12 Equivalence Rules that are possible. If we consider all those rules and apply them to our query there will be a huge number of plans that will be possible [14]. So the basic idea is to use these Equivalence Rules to find the alternate plans which are most cost efficient. We now list a number of general Equivalence Rules on relational-algebra expression where we use \( \Theta \), \( \Theta_1 \), \( \Theta_2 \) and so on to denote Lists of attributes, and E, E1, E2 and so on to denote relational-algebra expression.
i. Conjunctive Selection operations can be deconstructed into a sequence of individual selections.

\[ \sigma_{a_1 \land \ldots \land a_n}(E) = \sigma_{a_1}(\sigma_{a_2}(\ldots(\sigma_{a_n}(E)\ldots))) \]

ii. Selection operations are commutative.

\[ \sigma_{a_1}(\sigma_{a_2}(E)) = \sigma_{a_2}(\sigma_{a_1}(E)) \]

iii. Only the last operations in a sequence of Projection operations are needed, the others can be omitted.

\[ \Pi_{a_1} (\Pi_{a_2} (\ldots (\Pi_{a_n}(E)) \ldots)) = \Pi_{a_1}(E) \]

iv. Selections can be combined with Cartesian products and theta joins.

a. \[ \sigma_{a_1}(E_1 \times E_2) = E_1 \quad \text{if} \quad \sigma_{a_1}(E_1) \neq \emptyset \]

b. \[ \sigma_{a_1}(E \bowtie_{a_2} E_2) = E \bowtie_{a_1 \land a_2} (E_1 \bowtie_{a_2} E_2) \]

v. Theta join operations are commutative.

\[ E \bowtie_{a_1 \land a_2} E_2 = E \bowtie_{a_2 \land a_1} E_1 \]

vi. a. Natural-join operations are associative.

\[ (E_1 \bowtie_{a_2} E_2) \bowtie_{a_3} E_3 = (E_1 \bowtie_{a_2 \land a_3} E_2) \bowtie_{a_3} E_3 \]

b. Theta joins are associative.

\[ (E_1 \bowtie_{a_2} E_2) \bowtie_{a_3} E_3 = (E_1 \bowtie_{a_2 \land a_3} E_2) \bowtie_{a_3} E_3 \]

vii. The Selection operation distributes over the theta-Join operation under the following Conditions:

a. \[ \sigma_{a_1}(E \bowtie_{a_2} E_2) = (\sigma_{a_1}(E) \bowtie_{a_2} E_2) \]

b. \[ \sigma_{a_1 \land a_2}(E \bowtie_{a_2} E_2) = (\sigma_{a_1}(E) \bowtie_{a_2} \sigma_{a_2}(E_2)) \]

viii. The Projection operation distributes over the theta-join operation under the following Conditions:

a. \[ \Pi_{a_1 \land a_2}(E \bowtie_{a_2} E_2) = (\Pi_{a_1}(E) \bowtie_{a_2} \Pi_{a_2}(E_2)) \]

b. \[ \Pi_{a_1 \land a_2}(E \bowtie_{a_2} E_2) = \Pi_{a_1 \land a_2}(\Pi_{a_1 \land a_2}(E)) \bowtie_{a_2} \Pi_{a_2}(E_2) \]

ix. Set operations union and intersection are commutative.

\[ E_1 \cup E_2 = E_2 \cup E_1 \]

x. Set union and intersection are associative.

\[ (E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3) \]

\[ E_1 \cap E_2 \cap E_3 = (E_1 \cap E_2) \cap E_3 \]

xi. Selection operation distributes over the union, intersection and set difference operations.

\[ \sigma_{a_1}(E_1 \cup E_2) = \sigma_{a_1}(E_1) \cup \sigma_{a_1}(E_2) \]

xii. The projection operation distributes over the union operation.

\[ \Pi_{a_1}(E_1 \cup E_2) = (\Pi_{a_1}(E_1) \cup \Pi_{a_1}(E_2)) \]

We can use Multiple equivalence rules one after the other on a query or on parts of the query. \[15\] Query optimizers use minimal set of Equivalence Rules to reduce the number of ways an expression can be generated. A set of Equivalence Rules is said to be minimal if no rule can be derived from any combination of the others. We can also eliminate unneeded attributes by pushing projections based on Equivalence Rules. This results in reducing the size of the intermediate result. Given an expression if any subexpression matches one side of Equivalence Rule the optimizer generates a new expression where the subexpression is transformed to match the other side of the rule. This process continues until no more new expressions can be generated. If we generate an expression E1 from an expression E2 by using an equivalence rule then E1 and E2 are similar in structure and have sub expressions that are identical. This helps to reduce space requirement significantly.

We can utilize \[16\] Equivalence Rules to transform the query tree as follows.

a. Break up any Selection operation with conjunctive conditions into a cascade of Selection operations.

b. Move Selection operations as far down the query tree as possible.

c. Rearrange the leaf nodes of the tree so that most restrictive Selections are done first. Most restrictive Selections are the one that produces the fewest number of tuples.

d. Combine a Cartesian product with a subsequent Selection operation into a Join operation if the Selection condition represents a Join condition.

Other usage of equivalence rules is it plays a vital role in enhancing the performance of the query in the following ways:

i. It provides fast access for query processing and rule maintenance.

ii. It provides the ability to precompute all chains of inference that can occur with a query.

iii. It avoids delay during query processing.

iv. It enables query reformulation and modification that improves the cost of the query execution.

v. It examines the relative cost and helps to choose the least cost one.

vi. It helps to derive rules to operate anticipated queries.
vii) It helps in eliminating number of tuples that in turn reduces the cost.

viii) These rules can be used in systematic data analysis.

ix) It makes use of the domain values for attributes to recognize the redundant condition and discard the corresponding rules.

Thus Equivalence Rules are used to generate systematically all expressions equivalent to the given query which is more efficient to execute.

4. Cost Estimation

The main aim of query optimization is to choose the most efficient way of implementing the relational-algebra operations at the lowest possible cost. It is one of the difficult tasks in query optimization to accurately estimate the costs of alternative query plans. The cost of processing [17, 18] a query is expressed in terms of the total cost measure or the response time measure. The total cost measure is the sum of all cost components. Whereas the response time measure is the time from the initiation of the query to the completion of a query. The cost of a Query Execution plan includes the following components:

Access cost to secondary storage: This is the cost of searching for, reading, writing data blocks of secondary storage.

Computation cost: This is the cost of performing in-memory operation on the data buffer during execution.

Storage cost: This is the cost of storing immediate files that are generated during execution.

Communication cost: This is the cost of transferring the query and its result.

Cost estimation is of great importance in finding accurately and efficiently the cost of evaluating a query. [19] Cost estimation must be efficient since it is the inner loop of query optimization and it is repeatedly invoked. The cost of an operation is heavily dependent on its selectivity that is the properties of the input relations that forms the outputs. The basic requirement in estimating the cost of an operation is to collect statistical summaries of input and output data for a given each operator. Then derive the cost for each of its operators.

Once we have the costs for each of the operator nodes the cost for the plan can be obtained by combining the costs of each of the operator nodes in the tree. The cost estimation is mainly based on statistical information about the base relations and formulas to predict the cardinalities of the results of the relational operations.

A fundamental technique used in Cost estimation is cardinality estimation. [20] Optimizers take as input the cardinalities of tables at the leaves of a query tree and then use selectivity of operators in the tree to estimate the cardinality of the input to operators further up in the tree. To convert cardinalities to costs, Optimization techniques use Cost functions that estimate the cost of each Execution plan. Cost functions can be expressed with respect to either the total time or the response time. [21] A Cost function must be capable of optimize a given query in several dimensions. Cost function not only predicts the cost of operators but also formulas to evaluate the size of the result. The processing cost of query is evaluated in terms of the number of disk access I/O cost and CPU cost. The CPU cost is incurred when performing data operations in the main memory. The I/O cost can be minimized by using efficient buffer management technique. Cost estimation is an important factor that affects the performance of the query. So it is necessary to take proper measures to accurately estimate the cost of alternative Query plans.

5. Best Practices

We have suggested some of the practices to be followed to enhance the performance of the query.

a. We need to restrict the queries result set by selecting only the particular columns from the table rather than all columns from a particular table.

b. We should try to carefully use conditions in WHERE clause.

c. Try to avoid HAVING clause in SELECT statement.

d. Try to minimize number of sub query blocks within a query.

e. Try to use UNION ALL instead of UNION wherever possible.

f. Try to use operators like EXISTS, IN and JOIN appropriately in the query.

g. Try to use constraints for selection.

h. Avoid the use of view or replace view with original table.

i. Perform Select operation as early as possible to reduce the number of tuples.

j. Perform project operation as early as possible to reduce the number of attributes.

k. Select and Join operations that are most restrictive should be executed before other similar operations.

l. If we need to join several tables very frequently then we should consider create index on the joined columns.

m. Remove any unnecessary join from tables.

n. Ensure that multiple sub queries are in the most efficient order.
o. Try to use user defined function to keep the encapsulated code for reuse in future.

p. Try to use non-correlated sub query when dealing with large tables.

q. Try to use correlated sub query or derived tables for row-by-row operation on tables.

r. Try to drop indexes that are not being used.

s. Try to reduce the number of executions performed by selecting appropriate execution plan.

t. Try to make use of the references made to I/O and CPU cost to make the best decision.

u. Try to JOIN the tables using the most efficient join type and in the most efficient order.

v. We can make use of multiple equivalence rules to reduce the size of the intermediate result.

w. We can eliminate unnecessary attributes by pushing projections based on equivalence rule.

x. Try to use minimal set of equivalence rules to reduce the number of ways an expression can be generated.

y. If the costs of the operators are high then try to parallelize more sections of the query.

z. Try to share common subexpression so that space requirement can be reduced

6. Conclusion

In this paper we presented an introduction to basic aspects of Query Processing and Optimization. It is quite evident from the above discussion that Equivalence Rules play a vital role in the enhancement of Query Optimization. In this paper we primarily focus on the usage of Equivalence Rules in generating alternate equivalent efficient query and also some glimpses on the factors that can improve the query performance. Despite many research work has been carried out in this field of Query Optimization, significant problems still remain. However, from the above study we can understand that use of Equivalence Rules in generating alternate equivalent query strongly influences the subsequent performance of optimization. This study helps to make effective contribution in the area of Query Optimization. Additionally we have given some initial ideas for building modular query optimizer as part of an extensible database management system.

References


[17.]Jeffrey Sneiderman, Peter Pietzuch, Matt Welsh, Margo Seltzer and Mema Roussopoulos,“A Cost Space Approach to Distributed Query Optimization in Stream Based Overlays”,

Volume 2, Issue 6, June 2013

