Journal of Data Mining and Knowledge Discovery
ISSN: 2229–6662 & ISSN: 2229–6670, Volume 3, Issue 1, 2012, pp.-54-57.
Available online at http://www. bioinfo. in/contents. php?id=42

THE USE OF HINTS IN SQL-NESTED QUERY OPTIMIZATION

LOKHANDE A.D. AND SHETE R.M.

M. E Computer Science and Engineering Dept., Sipna C. O. E. T. Amravati, MS, India.
*Corresponding Author: Email- abhilokhande11@gmail.com and shete_rushikesh@yahoo.com

Received: February 21, 2012; Accepted: March 15, 2012

Abstract- The query optimization phase in query processing plays a crucial role in choosing the most efficient strategy for executing a query. In this paper, we study an optimization technique for SQL-Nested queries using Hints. Hints are additional comments that are inserted into an SQL statement for the purpose of instructing the optimizer to perform the specified operations. We utilize various Hints including Optimizer Hints, Table join and anti-join Hints, and Access method Hints. We analyse the performance of various nested queries using the TRACE and TKPROF utilities which provide query execution statistics and execution plan.
Keywords- Hint, query optimization, parsing.

Citation: Lokhande A.D. and Shete R.M. (2012) The use of Hints in SQL-Nested query optimization. Journal of Data Mining and Knowledge Discovery, ISSN: 2229–6662 & ISSN: 2229–6670, Volume 3, Issue 1, pp.-54-57.

Introduction

Query optimization is the most important stage in query processing where the database optimizer has to choose a query-evaluation plan with minimized cost and maximized performance [4,6]. The functionality of query optimization in deciding the best query execution plan is a significant task since there are a lot of possible queries in terms of the canonically equivalent algebraic expression for one given SQL query [4,7,8]. Moreover, the response time for those algebraic representations can vary one from another. Nonetheless, the efficiency of the chosen plan could make a great difference to the system's performance and hence this step should not be neglected. This is especially the case when the query being optimized consumes a considerable amount of resources such as physical disk reads and memory buffer. Thus, reasons such as these have made the task for query optimization critically important. There are three essential components of a query optimizer [4,9]: (1) search space, (2) cost modeling, and (3) search strategy. In the search space, there are several possible canonical equivalent query expressions who secriteria of each nested query type. The first condition is that the inner query block Q has a join predicate that references the outer query block. The second condition is that the SELECT clause of Q includes an aggregate function. The final condition is that the inner query block Q contains a division predicate. There are two commonly used query optimization techniques which are: the Heuristics and the Cost estimation technique. A Heuristics method transforms the initial query presentation into an efficient query tree equivalent based on the order of operation for executing a query [6,8]. Based on these classifications, Kim [4] proposed sets of algorithms to transform the initial nested queries to their canonical equivalent non-nested form. This allows the query optimizer to choose a merge join algorithm when implementing nested queries rather than applying the high cost nested-iteration method. Due to the estimation nature of this technique, the need for Hints has emerged. The introduction of Hints for nested query optimization has indicated the inability of those commonly used optimization techniques to produce the best strategy for every nested query. By applying Hints as an optimization technique, the search strategy is directed to choose the optimize execution plan based on the specified knowledge which is stated in Hints [2,]. There are two main goals of Hints optimization: a faster response time, and lower resource consumption. Moreover, the Hints approach is easy to

understand in terms of human language and its process is simple to follow 3.

Hints: a background and classifications

we use USE_HASH Hint to optimize the nested query between cust_order and staff tables. Oracle provides an extensive list of Hints that can be used to tune various aspects of a query performance [1]. We categorize Hints for nested query optimization into three parts: (1) optimizer Hints, (2) table join and anti-join Hints, (3) access method Hints.

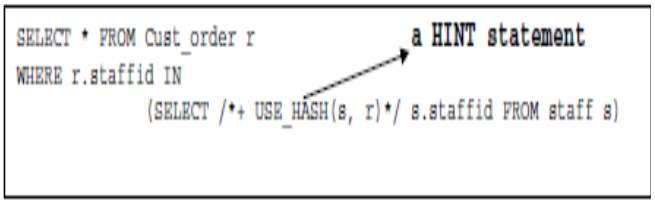

Fig. 1- Using a hint in an SQL statement

Optimizer Hints

Optimizer Hints apply a different optimizer to a query, which in turn redirects the overall processing goal The all_rows Hint explicitly chooses the cost estimation technique to optimize a statement block with a goal of best throughput [1] Oracle offers four kinds of optimizer Hints, which are all_rows, first_rows, rule, and choose The choose Hint causes the optimizer to choose between the Heuristics optimization and cost estimation techniques for an SQL statement. The optimizer bases its selection on the presence of statistics for the tables accessed by the statement.

Table join and anti-join Hints

There are several Hints in the table join and anti-join Hints provided by Oracle as shown in Fig. 3, which are: use_nl Hint, use_merge Hint and use_hash Hint. The use_nl Hint causes Oracle to join each specified table to other row sources with a nested loop join that uses the specified table as the inner table. In general, the default join method in Oracle is the nested loop join [24]. Therefore, use_nl Hint is not used in the performance evaluation. There are six Hints which can be used for join nested query optimization:

- The use_hash Hint. The Hint invokes a hash join algorithm to merge the rows of the specified tables. In many circumstances, it changes the access path to a table in addition to the change in join method [12].
- The use_merge Hint. Alternatively, the Hint forces the optimizer to join tables using the sort-merge operation [2].

```
select null
from staff s
where s.city not in (select c.city from customer c)

Rows        Execution Plan
----------  ---------------------------------------------------
       0    SELECT STATEMENT   GOAL: RULE
     497      FILTER
    1011        TABLE  ACCESS  GOAL:  ANALYZED (FULL) OF 'STAFF'
      77        TABLE  ACCESS  GOAL:  ANALYZED (FULL) OF 'CUSTOMER'
```

(a) NO HINT

```
select /*+ FIRST_ROWS */ null
from staff s
where s.city not in (select c.city from customer c)

Rows        Execution Plan
----------  ---------------------------------------------------
       0    SELECT STATEMENT   GOAL: RULE
     497      FILTER
    1011        TABLE  ACCESS  GOAL:  ANALYZED (FULL) OF 'STAFF'
      77        TABLE  ACCESS  GOAL:  ANALYZED (FULL) OF 'CUSTOMER'
```

(b) with HINT

Fig . 2- The effect of adding an optimizer hint

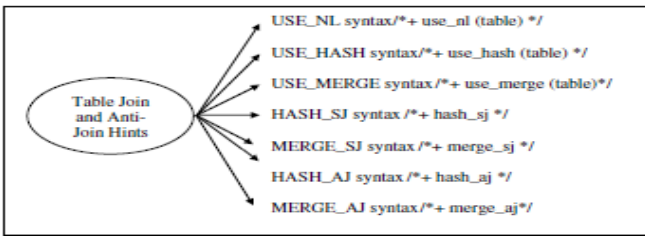

Fig. 3- Table joins and anti-joins hints

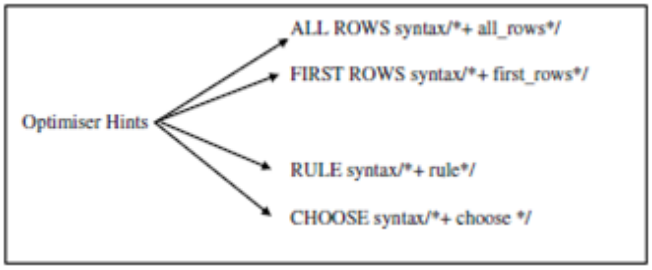

Fig. 4- Optimizer hints

The hash_sj Hint. This is a semi-join Hint that is added to the inner block of a correlated nested query using the EXISTS clause for the purpose of transforming the query into a hash semi-join to access the specified table.

- The merge_sj Hint. Similarly, the Hint also transforms a correlated EXISTS nested query into a semi-join except that it invokes a merge semi-join.
- The hash_aj Hint. The Hint is placed in the inner loop of a NOT IN nested query to invoke a hash anti-join operation.
- The merge_aj Hint. The Hint joins the tables in a NOT IN nested query using the merge anti-join operation.

Experimental Results

For the purpose of experimentation in the paper, a Sales database has been created as a sample database. We perform the experiments on Pentium IV 1. 8 GHz CPU with 256MB. We use Oracle9i Database to perform our SQL queries. The Exception is given to table Cust and table Staff; in our experimental results the terms Cust and Customer are interchangeable to identify table Cust; also in our experimental results we use name Staff_S and Staff interchangeably to identify table Staff.

In the next section, we use an experimental Hints optimization technique and also an optimization technique without using Hints. For the No-Hint approach, the optimizer system will choose the commonly used optimization techniques which are: the Heuristics and Cost estimation techniques. We also compare those optimization techniques results based on the join nested query framework

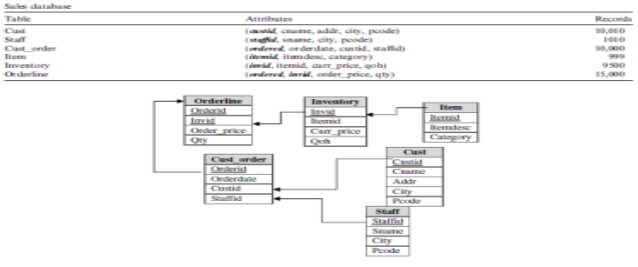

Fig. 5- Sales order systems

Access method Hints
The access method Hints can be classified into Hints without index and Hints with index. Rowid Hint is one of Hints without index with syntax: /*+ Hint(table) */. Hints with index consist of index Hint, index_ join Hint and index_ffs Hint with syntax: /*+ Hint (table, index) */. As shown on Fig. 6, there is a SELECT statement with index Hint consist of two parameters which are V as the alias of table Inventory and Inventory_invid_pk as the table index of table Inventory.

Nested query optimization framework
In this section, we explain the nested query and the optimizer modes used in our experimentations.

Nested queries
Basically, a nested query is a query within another query in a nested form and can be classified into four groups [6,7,13], which are:
- Standard nested queries (IN),
- Non-equality nested queries (ANY/ALL with conditional operators),
- Semi-join nested queries (EXIST),
- Anti-join nested queries (NOT EXIST and NOT IN).

Optimizer modes
There are two kinds of optimizer modes in Oracle: the Heuristics optimizer and the Cost estimation optimizer [6]. The Heuristics optimizer's main function is to reduce the size of the intermediate result. In general, the SELECT and PROJECT operations, which reduce the number of records and the number of attributes respectively, are found to be the most restrictive operations [6]. Apparently, the re-ordering of the leaf nodes on the query tree can have a significant impact on the system's performance and, hence, good Heuristics should be applied whenever possible. However, as shown in the performance evaluation section, this optimizer mode is less effective on many join nested queries optimizations when compared with using Hints on CBO optimizer mode. The Cost estimation optimizer determines which execution plan is the most efficient by considering available access paths and by factoring in information based on statistics for the schema objects (tables or indexes) accessed by the SQL statement [11].
For instance, the decision to choose between an index scan and a full-table scan for a table with skewed (disproportional) data will not always be obvious, and hence cannot be concluded without first examining the statistics. A Cost estimation optimizer will favor an index scan over a full-table scan if a given query retrieves less than 40% of the data in a table. On the other hand, a full-table scan will be preferred if the retrieved data will be over 60%. As can be seen, the Cost estimation optimizer is more flexible and intelligent in choosing the SQL operations. Therefore, it should make a better execution plan than the Heuristics optimizer does if it has all the statistical information it needs. Moreover, Cost estimation has two main optimization goals: the fastest response time with minimum resource usage, and the best throughput with minimum total resource consumption.

Performance evaluation
In this section, we present our performance evaluation result incorporating Hints using TKPROF utility in Oracle.

5. 1. Execution plan (query tree) and TKPROF utilities
In general, an execution plan is the strategy produced and used by the optimizer to process a query in order to get the desired result [12]. It is also known as the query tree due to its tree-like structure. It is essential to understand the functioning of the execution plan, as it provides a useful indication of the performance of the query processing. For instance, as shown in the following illustration (Fig. 10), an indexed nested loop join is used to join the tables in the given query. The join method may indicate an efficient processing, as the previous section explained that the presence of an index in the inner table may dramatically improve the performance of the nested loop join. However, the operations used in the execution plan are only an indicator, not an absolute answer. Moreover, its main function is to explain the steps involved in the processing and the sequence of execution. As can be seen, there are four steps involved in the processing of the given query:
Step 1: Nested loops;
Step 2: Full-table access to Orderline table;
Step 3: Table access to Inventory table by index ROWID;
Step 4: Index unique scan on the primary key of the Inventory table.
The operations below the nested loops are indented indicating that they are executed prior to the join operation. Hence, the first execution is Step 2, which will retrieve a row from the Orderline table and return it to

Step 1. Then, Step 1 will send the invid (inventory number) to Step 4 for every row obtained from Step 2.
Step 4, the optimizer will perform an index unique scan on the invid to gather the ROWID. If no ROWID is found, Step 3 will instruct Step 1 to discard the row. Otherwise, Step 3 will use the ROWID to access the Inventory table to retrieve the corresponding row. Finally, Step 3 will return the row to Step 1, where the row will be joined with that from Step 2 if the current price is equal to the order price. If the two prices are not the same, Step 3 will not return a row, and Step 1 will eliminate the row from the result set. The steps will be carried out recursively until all the rows have been processed. The TRACE utility is also known as the SQL trace facility, which can be activated only by setting the relevant parameters. There are two types of parameters that need to be carefully set [2]: a file size and location parameters (eq. User_dump_dest=c:norahomentracefile) b function enabling parameters (e. g. time_statistics and sql_trace). However, the report produced is difficult to understand. Therefore, it must be used in conjunction with the TKPROF utility, whose task is to translate the traced result into a readable format [2]. When the Oracle database is installed, TKPROF is automatically installed in the directory norahomenbin. Firstly, we add norahomenbin to the current path. Then, the utility is invoked by using the following command:

Tkprof<TraceFile><outputFile>[explain=<username><password>]

The trace file has an extension of . trc, while the output file that is in the readable format will have an extension of . prf. It is essential to specify the location in both the hTraceFile iand the hOutputFile i, otherwise, TKPROF would not be able to perform its task due to the missing file. The following shows the command line we use to

invoke TKPROF: Tkprof d:\ora0001. trc d:\trace. prf explain = system/manager. As shown in Fig. 11, the TKPROF result is divided into three parts which are: sql statement, statistic information and execution plan. The statistics section contains seven statistics measurements and one call column (the first column) indicating the phases of query execution. The following will provide a description of the phases and the statistics measurements.

Phases

• Parse. The parser checks the syntax and the semantics of the SQL statement, which will be decomposed into relational algebra expressions for execution.

• Execute. After the parsing and validation stage, the optimizer will execute the decomposed SQL statement using the execution plan operations shown in Fig. 11.

• Fetch. Finally, in the fetching phase, the optimizer will retrieve the final output from the corresponding tables based on the instruction passed from the query execution phase.

Conclusions

The need for a query optimization technique for providing the best response time and the best throughput in query processing is very important. Considering that nested queries are the most complex and expensive operators, this paper has focused on providing an effective optimization technique by using Hints. The chosen technique is precisely a manual tuning that requires users to insert additional comments into an SQL statement.

Acknowledgement

I express my sincere gratitude to Resp. Dr. A. D. Gawande Head of the Department, Computer Science & Engineering & Resp. Prof S . Dhande for providing their valuable guidance and necessary facilities needed for the successful completion of this seminar throughout. I am also obliged to our principal, Resp. Dr. S. A. Ladhake who has been a constant source of inspiration throughout.

Lastly, but not least, I thank all my friends and well-wishers who were a constant source of inspiration.

References

[1] Andrei M. and Valduriez P. (2001) *User-optimizer communication using abstract plans in Sybase ASE*, 29-38.

[2] Burleson D. K. (2001) *Oracle High-Performance SQL Tuning*.

[3] Chaudhuri S. and Shim K. (2003) *An overview of cost-based optimization of queries with aggregates, IEEE,* 18(3) 3-9.

[4] Chaudhuri S. (2006) *An overview of query optimization in relational systems*, 34-43.

[5] Chaudhuri S. and Shim K. (2009) *Optimization of queries with user-defined predicates*, 24(2), 177-228.

[6] Elmasri R. and Navathe S. B. (2004) *Fundamentals of Database Systems*.

[7] Ganski R. A. and Wong H. K. T. (2005) *Optimization of nested SQL queries revisited*, 23-33.