THE OPTIMIZATION OF QUERIES IN ECONOMIC APPLICATIONS THAT USE RDBMS'S

PhD. Danut - Octavian SIMION
"Athenaeum" University of Bucharest, Romania
danut_so@yahoo.com

Abstract:
The paper presents the advantages of the optimization of queries in economic applications that use RDBMS's. The economic applications that are based on a relational database management system (RDBMS) often require fast responses at the queries made by users. Fast responses mean a good design of the database and saves time and money for the end users and for companies that use these types of applications. Sometimes a good design doesn't always ensure fast responses of queries so other methods of optimization may improve time and enables users to obtain more from their economic applications. There are methods to optimize queries such as transforming the queries using algebraic rules, modifying the initiation parameters of RDBMS, using hints for optimization, using instruments such as Automatic SQL Tuning. This optimization needs to analyze queries and to make an optimization plan by using explain plan command to calculate statistics and the most important task is to create the appropriate indexes for one or more columns. These methods helps the users to get fast responses for their queries and to improve the design and rules for the database that they use in specific economic applications.

Keywords*: SQL Tuning, Economic applications, RDBMS, PL/SQL, optimization hints, indexes, UML diagram.*

Introduction

Applications retrieval data is expressed in different relational languages for RDBMS. To obtain optimum results, use automated interfaces rewrite retrieval requests by taking two steps:

• Expression retrieval requests in the form of relational algebraic expressions, which is based on the equivalence between calculation and relational algebras;

• Applying transformations on relational algebraic expressions constructed in the previous step, to obtain equivalent and efficient relational expressions.

The transformation can be achieved by two optimization strategies: general and specific. General strategies are independent of data storage mode. They are based on properties of relational algebra operations (commutability, associativity, composition). Such strategies are selected before the junction, before the junction projection, selection before the screening and combining multiple selections. Specific strategies take into account the data storage module and are characteristic of an RDBMS. The elements that influence the execution of operations that occur in a retrieval request are: direct access ordering rules algebraic expressions specific to a RDBMS [2], [5]. There are some methods to optimize queries by modifying initialization parameters or by using optimization hints or by using specific/graphic instruments, but the most important is to use an explain plan to calculate statistics and to create indexes or to transform/rewrite the query.

1. Using initialization parameters for optimization

Many boot parameters are used to adjust and increase database performance. For example DB_FILE_MULTIBLOCK_READ_COUNT defines the number of blocks read simultaneously accessing a database table. It is used to optimize the total browsing a table when looking for a certain value of a column in the corresponding row.

The parameter SQL_TRACE - write to a trace file for SQL executed and the statistics includes information about:

- Number of parsing, execution and results data sets;
- Different CPU execution time;
- The number of physical and logical reads;
- The number of rows processed;
- Number of library cache omitted;

2. Create an execution plan for queries

A good idea for optimization is to use an EXPLAIN PLAN command that can be included to explain preordered SQL execution plan. Execution plan is a sequence of physical operations that RDBMS must execute to return the required data. By analyzing the execution plan for SQL queries, it can be seen which of them are ineffective and can compare the alternatives to find out which can give better performance. An example of the command is:

```
SELECT LPAD(` `, 2*level) || operation ||' ` ||
         options ||' ` || object_name "Execution Plan"
  FROM plan_table
CONNECT BY PRIOR id = parent_id
```

```
  START WITH ID = 1;
```

This command will have as result the following:

```
SELECT STATEMENT
    NESTED LOOP
         TABLE ACCESS          BY ROWID          table1
           INDEX               RANGE SCAN        column_idx1
         TABLE ACCESS          BY ROWID          table2
           INDEX               RANGE SCAN        column_idx2
```

The analysis of this data shoes if indexes are used, if the attributes which make binding tables (join) are indexed, or whether a particular table is accessed in full scan mode or RANGE SCAN. Following these analyzes, the programmer / database administrator decides to create new indexes or SELECT statements hint will be introduce to use existing indexes [1], [6].

The command ANALYZE enable validation and calculation of statistics for an index, table or cluster. These statistics are used by the optimizer based on the cost when calculating the most efficient plan for data mining. In addition to its role optimizer, ANALYZE also helps to validate objects and manages space structure system.

An example of this command is:

```
ANALYZE TABLE table1 COMPUTE STATISTICS;
```

It is also possible to calculate statistics with following command that can be used to estimate 50% of rows:

```
ANALYZE TABLE table1 ESTIMATE STATISTICS SAMPLE 50
PERCENT;
```

Other RDBMSs provides a procedure that allows the user to analyze a scheme:

```
EXECUTE
DBMS_UTILITY.ANALYZE_SCHEMA(`USER1','COMPUTE');
```

In this command USER1 is the owner of tables, clusters and indexes, and COMPUTE is the type of the analysis that is required.

3. Using optimization hints for queries

Optimizing hint can be used with SQL commands to modify the construction plans. Hint allow the programmer to make decisions for the optimizer, so a hint mechanism may instruct the optimizer to choose a

specific execution plan for a query based on certain criteria. For example, the programmer may decide that a certain index is more suitable for different queries. Based on this information hints instructs the optimizer to use the optimal execution plan [3], [4].

Hint sites can be used for the following types:

• Single tables - hints specified for a table or view. INDEX and USE NL are such hints;

• Multi-tables - hints are similar for single tables, except that the hint can specify multiple tables or views. LEADING hint is an example of multi-table. USE_NL (table1 table2) is not considered multi-table hint because it is an abbreviation for USE_NL (table1) and USE_NL (table2);

• Query Block - hints for block query sites operate on a single block or more. STAR TRANSFORMATION and hint UNNEST are examples of query block sites;

• Statement - hints statement applies to the entire SQL statement. ALL ROWS is an example of a hint.

Hints to transform Queries (Query Transformations)

Each of the following hints instructs the optimizer to use the following SQL query transformations:

- NO_QUERY_TRANSFORMATION
- USE_CONCAT
- NO_EXPAND
- REWRITE
- NO_REWRITE
- MERGE
- NO_MERGE
- STAR_TRANSFORMATION
- NO_STAR_TRANSFORMATION
- FACT
- NO_FACT
- UNNEST
- NO_UNNEST

Hints to order connections/relationships between tables/views (Join Orders)

The following hints suggest join order:

- LEADING
- ORDERED

Hints for Join Operations (Join Operations)

Each of the following hints instructs the optimizer to use a specific join for a table:

- USE_NL
- NO_USE_NL
- USE_NL_WITH_INDEX
- USE_MERGE
- NO_USE_MERGE
- USE_HASH
- NO_USE_HASH

Use of hints USE_NL and USE_MERGE are recommended with any other hints join order. The RDBMSs uses these hints when referenced table is forced to be the inner table in a join.

When used, hints may use a whole set of rules to ensure optimal execution plan. For example, in the case of complex queries with multiple relationships between tables, and when is specified only the INDEX hint for a given table, the optimizer must determine the remaining access paths for use, and appropriate methods for join. Therefore, even if the INDEX hints are given, the optimizer might not necessarily use that hint, because it determines the required index that cannot be used due to the join methods and access roads. In the following example, the LEADING hint specifies the exact order's join (relationship); join methods to be used are also specified [1], [4].

```
SELECT /*+ LEADING(t2 t1) USE_NL(t1) INDEX(t1 col1_id_pk)
           USE_MERGE(t3) FULL(t3) */
    t1.column1, t1.column2, t3.column_id, sum(t2.column3)
total_col3
  FROM table1 t1, table2 t2, table3 t3
  WHERE t1.column1_id = t2.column1_id
    AND t1.column2_id = t3.column2_id
  GROUP BY t1.column1, t1.column2, t3.column_id
  ORDER BY total_col3;
```

By using the structure's hint global view can change and it can be avoid specifying the index hint in the body's view. To force the use an index in a table it can be specified one of the following commands:

```
SELECT /*+ INDEX(v.t2.t3 col_ix) */  *
  FROM v;

SELECT /*+ INDEX(@SEL$2 t2.t3 col1_ix) */ *
  FROM v;

SELECT /*+ INDEX(@SEL$3 t3 col2_ix) */ *
  FROM v;
```

4. The usage of a SQL tuning advisor

Automatic SQL Tuning capabilities are provided by a tool called SQL Tuning Advisor server. SQL Tuning Advisor takes one or more SQL statements as input and Automatic Tuning Optimizer is invoked to perform tuning commands for SQL statements. The result for SQL Tuning Advisor is in the form of advice or recommendation, along with a rationale for each recommendation and the expected result. Recommendation refers to a collection of statistics on objects, creation of new indexes, restructuring orders SQL statements, or creates new profiles. A user can choose whether to accept the recommendation to complete the tuning SQL commands. Inputs for SQL Tuning Advisor can be for a single SQL command or set of commands SQL Tuning Set to be first created. An STS is a database object that stores multiple SQL commands with execution context. An STS can be created manually using an API command line interface or through a framework [3], [5]. Recommended interface for running SQL Tuning Advisor tool is a specific framework of the RDBMS. SQL Tuning Advisor can be run using DBMS_SQLTUNE package. To use this API, the user must grant specific privileges and follow the following steps:

- Creating a SQL Tuning Set (if tuning multiple SQL commands include)
- Create a SQL tuning task
- Running a SQL tuning task
- Showing results for SQL tuning task
- Implementation of the recommendations

Creating a SQL Tuning Set.

It can be created a tuning task in the text of a SQL. SQL Tuning Set is containing multiple commands, a SQL command selected. SQL identifier of the cursor cache or a statement of Automatic Workload Repository for selected SQL. For example, SQL Tuning Advisor use a SQL commands necessary to optimize it must be passed as a CLOB argument. For example:

```
DECLARE
 my_task_name VARCHAR2(30);
 my_sqltext   CLOB;
BEGIN
 my_sqltext := 'SELECT /*+ ORDERED */ * '
||
               'FROM table1 t1, table2 t2, table3 t3 ' ||
               'WHERE t1.col1_id = t2.col1_id AND '  ||
                     't2.col2_id = t3.col2_id AND '      ||
                     't.col2_id < :bnd';

 my_task_name := DBMS_SQLTUNE.CREATE_TUNING_TASK(
```

```
         sql_text    => my_sqltext,
         bind_list   =>
sql_binds(anydata.ConvertNumber(500)),
         user_name   => 'HR',
         scope       => 'COMPREHENSIVE',
         time_limit  => 30,
         task_name   => 'my_sql_tuning_task',
         description => 'Task to tune a query on a specified
employee');
END;
/
```

In this example, 500 is the value for the variable binding (bind): bind passed as a function argument type for SQL_BINDS. The function CREATE_TUNING_TASK analyze user SQL command and the goal is to set COMPREHENSIVE representing that performs the analysis adviser SQL profiling and 60 seconds is the time spent on the run. In addition values are available for the task name and description. After creating the task tuning, it must be executed to start the tuning process.

```
BEGIN
  DBMS_SQLTUNE.EXECUTE_TUNING_TASK( task_name =>
'my_sql_tuning_task' );
END;
/
```

5. Other methods used to optimize queries

There are other methods of retrieval query applications derived from how to build a relational database and the organization of data and acting on their orders:

- Use rules of E.F. Codd in the design of a database;
- Use and implementation of normal forms (NF1 - NF5) when designing / redesigning tables and their relationships;
- Use indexes to tables and columns used in relationship to those used in WHERE clauses;
- For complex queries that contain sub-queries are recommended intermediate tables, so data can be indexed and made available more quickly;
- To optimize the SQL, it is recommended to include them into objects stored in the database (functions, procedures, packages);
- For frequently used queries and views are recommended temporary tables in memory (global temporary table) or a special data type vector (array, array of tables, etc.);
- For complex retrievals are recommended tools for data mining, OLAP, ROLLUP, CUBE - specific data warehouse (data warehouse);

- Use EXISTS clause instead of IN clause in a complex query that includes a sub-query;
- It is recommended to avoid data transformation functions (trim, TO_CHAR, TO_DATE, TO_NUMBER, etc.) for attributes used in WHERE clauses because their associated indexes are no longer used;
- It is recommended queries using two or more tables to the FROM and WHERE clauses use (JOIN conditions) order from low to high (less data tables - tables for more data);
- Use the BULK COLLECT clauses for particular types of data;
- Use Dynamic SQL Statements (written as text) using bind variables and because they are parsed before being executed by special DBMS_SQL package;
- Use explicit cursors (declared by programmers), because allow control activities of transactions, the number of rows (tuples) brought, etc.;
- It is recommended rewriting SQL commands by keeping the column order in the SELECT phrase corresponding tables in the FROM clause, use of aliases, etc.;
- In SQL statements - SELECT applying computing functions such as COUNT, use COUNT expression (number) instead of COUNT (*);
- In complex sentences SQL - SELECT using sub-queries in the FROM clause is recommended to use the expression FROM (SELECT column1, column2 FROM ....);
- In terms of administration it is recommended increasing BUFFER segment, cache, temporary table space and deal with situations that had an error, warning sites, jobs executed, etc.;
- It is recommended that database administrators to use tools in the enterprise resource management activities, to discontinue the remaining sessions hung consuming large resources of space (TABLESPACE) and CPU;
- It is recommended to run large job sites or those data backups to run at times when there is little time users connect to the database;
- It is recommended tuning database through its configuration parameters, taking into account the recommendations and documentation for specific RDBMS [2], [4].

6. Conclusions

To optimize retrieval applications in RDBMS's is necessary to take into account the recommendations of the manufacturer to be consulted specialized forums, use of the service request type and not least to be taken into consideration basic rules used when designing the database. It is also useful to use EXPLAIN PLAN SQL commands as appropriate, ANALYZE, HINT SQL and SQL Tuning

Automatic Tool Advisor. The optimization work involves both programmers who write SQL and using development frameworks, and database administrators using specific data management tools (management console, database parameter setting, tuning, etc.) [1], [6]. Optimizing queries is an important task in economic application and these activity allows users to extract more rapid the information they need and so the efficiency of those increase in a significant way. A good design of the database is required and also to use different tools or hints to increase efficiency of queries. Also using specific algebra rules helps in the join operations between algebraic sets, by creating and using different indexes on specific columns [2], [4]. Economic applications can benefit from an efficient way of using queries that are created by programmers or tuned by the database administrators based on a good design and usage of UML diagrams.

References

1. Dan Tow, "*SQL Tuning*", O'Reilly, 2012;
2. Donald K. Burleson, "*Advanced Oracle SQL Tuning*", Rampant Teo-Press, 2013;
3. Kevin S Goff, "*Productivity Tips for Optimizing SQL Server Queries*", Code Magazine, 2014;
4. Rodrigo Koch, "*SQL Database Performance Tuning for Developers*", Toptal, 2012;
5. Pinal Dave, "*SQL SERVER – Tips for SQL Query Optimization by Analyzing Query Plan*", Blog SQL Authority, 2013;
6. Sean McCown, *"7 performance tips for faster SQL queries*", InfoWorld, 2012;

URI: http://technet.microsoft.com/en-us/library/
URI: http://msdn.microsoft.com/en-us/library/ff650689.aspx
URI: http://dev.mysql.com/doc/refman/5.6/en/
URI: http://hungred.com/useful-information/