Investigation in Recommendations for Troubleshooting of Ineffectiveness of Queries to Database Tables

Ali Mohammed Saleh Ahmed

Electronic Computer Center, University of Diyala, Diyala, IRAQ

dr.alisaleh80@gmail.com

Submission date:- 23/6/2018 Acceptance date:- 4/7/2018 Publication date:- 12/12/2018

Keywords: DBMS, SQL-instructions; optimization of applications; Database, Oracle.

.

Abstract

One of the ways to increase work output of the applications connected with database is optimization of queries to database tables**.** Importance of query optimization issue increases with growth of records number to decades and hundreds of thousands tables which the query is referring for. There are various recommendations on troubleshooting of queries performance deceleration. However, there are no results of experimental investigations on effectiveness of applying other recommendations given in scientific literature. The authors have conducted numerical experiments for investigation of recommendations, both existing and proposed by the authors, to determine the quantitative values of effectiveness of applying recommendations. A number of recommendations are illustrated by means of examples of applying recommendations including several alternative variations. The work describes differences in effectiveness of applying recommendations in different systems of database management considered in this article. The work describes relative values of effectiveness of applying recommendations compared with source queries. As a result of article, the authors have developed the algorithm of query text analysis and applying recommendations. Investigation results expand knowledge on practical applying both known and new recommendations for optimization of queries to database tables. The article has practical orientation and it is methodically useful for first time programmers in a field of structured queries language.

1- Introduction

During tracking of information system a number of troubles arise. [1]. One of these troubles is increasing of records number and database capacity, as a result of which the time of query performance becomes intolerable. Among the other methods of application work acceleration, we will consider the methods of query optimization carried out on database management system (DBMS).

2- Common issues for queries optimization

Optimization of queries of relation databases consists of two parts equal by their aspects importance. The first one does not depend on developer; is an inner mission of the DBMS, the purpose of which is determination of the most effective way to carry out the query. The second aspect completely depends on developer skills because it consists

of writing such queries for which the DBMS could be able to use the most effective ways for data detection. The feature of queries written in declarative language involves the fact that to achieve the similar result completely different types of queries it can be written consisting of different operators, with various ways of tables splitting, differently using the indexing advantages etc.

There are different recommendations for increasing of queries writing quality [1-10]. However, the authors indicate the quantitative values of effectiveness of these recommendations using. There are many questions. What algorithm programmer should follow to analyze the query for its optimization by the performance time? Are the clauses recommendations equally operating for each DBMS? This article is devoted to answers for all these questions.

One of the key optimization terms is a query plan. Query plan is methods for algorithm of their by means of which the DBMS is able to obtain the desirable result. After lexical and syntax analysis of the query text its optimization is conducted, and then the DBMS chooses the most relevant plan and carries it out.

That is to say that all developers have different approaches for various DBMS for issue of index organization. There are two types of indexes provided for DBMS MS SQL Server: clustered (cluster) and non-clustered (non-cluster). Table data is stored in sorted form only if a clustered index is created for this table. Table which has no clustered index is called stack. The Firebird SQL type or Oracle type DBMSs does not support creation of clustered indexes; nevertheless, they support creation of non-clustered indexes[7].

To determine the effectiveness of possible query modifications the tests were made on database. The database subject to analyzing is supported by the database management system Firebird SQL. The amount of information consists of more than 10 gigabytes, and the amount of tables is equal to several hundreds of existence the database has accumulated about million records, therefore the issue of organizing quick data access and transparent logic of SQL-instructions are still relevant.

Original database of the enterprise is supported by Firebird SQL, but for comparative analysis of some recommendations the structure of original database has been created in MS SQL Server.

3- Causes of queries ineffectiveness and troubleshooting recommendations

Let’s consider the causes of queries ineffectiveness, troubleshooting recommendations and their effectiveness:

1- Synonyms are not used for the tables.

The synonyms for the tables should be applied in all queries where more than one table is mentioning in a clause FROM. Though using of synonyms in such queries insignificantly accelerates detection operation of SQL by operator by means of DBMS core due to decreasing of queries recursion; nevertheless, it improves readability and relief the code detection by the developer. Effectiveness of this recommendation is 1.42%.

2- Improper order of conditions listing in FROM section.

The order of tables listing in a FROM section is relevant for optimizer when he is searching the way to carry out the query. Optimizer is working as follows: firstly, he considers clauses “WHERE” and gives all tables a determined weight based on a predicate type, then he chooses the table with the least price and makes it the master table. However, there is one nuance which is that if several tables obtain the same value, and it is the least, then the optimizer chooses the table as the last table in the “FROM” clause. Therefore, the last in the list must be the table that returns the smallest number of rows[5].

During the experiment, a query was selected in which two tables containing different number of records according to the conclusions of the DBMS optimizer, has obtained the same price of performance. To find out the truth of this recommendation, the query was performed in two variations: one of two tables, which are equal in value, returning the smallest number of records, was specified before the table that returns the largest number of records and vice versa. The query in which the table returning the smallest number of rows was the last, has been run by 932 ms faster, the effectiveness of this recommendation was 9.89%.

Thus, after the preliminary query plan is formed, it is necessary to look at the prices of the tables participating in the query. In the case that for several tables the price is the same and the query requires faster performance, the table that returns the lowest number entries must be indicated at the end of the "FROM" section.

3- Query consisting of the operators EXIST (IN) and DISTINCT is periodically delaying.

Effectiveness of the operators EXISTS and IN depends on number of rows, returned by each query.

In a query with IN, the manage table is the sub query specified in IN, the master query is repeating for each row. In a query using EXISTS, on the contrary, the manage table is the main query, and the sub query specified in EXISTS is repeated for each row that is selected in the main query.

Thus, if the sub query returns a small amount of rows and the main query returns the big amount of rows it is better to use operator IN. Otherwise, it is necessary to use the operator EXISTS.

However, there are some particular cases when it is necessary to find the first inverted row by the value of any field and you can achieve the same result by means of the operator DISTINCT. In a database such as Firebird SQL or Oracle, the DISTINCT statement will cause the database engine to look through all the records from the required table one by one, in the database organized by MS SQL Server, the clustered index will be viewed, if any, or a full table scanning will be performed if the index does not exist, until the desired value is met. In the case when the search value is in the first rows of the table, the speed of finding the line will be extremely high, but in the case where the search string is near the end of the table, the search will be complicated. And in this case, the use of EXIST or IN operators will be more efficient than using of operator DISTINCT[9].

Recommendation: query must be divided in two parts by means of operator EXIST or IN. In the course of the experiment, the effectiveness of this recommendation was 95.04%.

4- Extra use of operators for the line set operation.

To apply of the Boolean operations for line set (subtraction, merging, addition and so on) it is necessary to represent the structure of the base clearly. When using sub queries, it is necessary to have an idea of the approximate number of records returned by each sub query. In the most cases, with the competent use of operations on sets of lines, you can significantly increase the efficiency. For the tutorial database, the efficiency of excluding one "EXIST" operator as well as revision of the set of line relationships and modification of the query text, led to an increase in efficiency for 97%.

5- Combination of the approved amendments in the table with the absence of necessary indexes.

One of the slowest commands in SQL is an UPDATE command because the most part of approved amendments in the tables requires the complete review of the tables. As a result, these operations are resource intensive and too slow when the tables are very huge. The trouble of low productivity is arising because there is no information used in query from the table to restrict the amount of current rows [7]. If any term of the WHERE clause of the UPDATE operator will not separate the rows set, the UPDATE operator requires a huge amount of processing time.

The ways of extra lines separating could be various. Let's consider the layout (Listing 5-1) of a typical query UPDATE. The records number in the table ABONENT1 is significantly more than the records number in the table ABONENT2.

(Listing 5-1) – Original query UPDATE

```
UPDATE ABONENT1
    SET VAL1 = (SELECT F1
                FROM ABONENT2
                WHERE ABONENT1.ID = ABONENT2.EID)
```

When performing this query, the DBMS core will need to review all records from the table ABONENT1, and, in the case if ID of the subscribers is similar from two tables, the field VAL1 will be refreshed. While among the amount of records from table ABONENT1 even those which clearly do not satisfy the term will be attempted to be updated, this entail slow performance of the query. Therefore, modifications of the original query are possible in which extra lines will be deleted.

The first variation of query modification (Listing 5-2) with a using of "IN", which effectiveness was 24.11%.

(Listing 5-2) – Modification of UPDATE using IN

```
UPDATE ABONENT1
    SET VAL1 = (SELECT F1
                FROM ABONENT2 A2
                WHERE A2.FID IN (SELECT
                A3.ID
                                 FROM ABONENT2 A3
                                 WHERE A3.FID = ABONENT1 .ID
```

However, it is possible to modify this query by means of the operator "EXIST" (Listing 5-3). The benefit of this modification is the ability of operator "EXIST" to find the accordance between small amount of records in the table ABONENT2 and big amount of records in the table ABONENT1.

(Listing 5-3) - Modification UPDATE assisted by EXIST

```
UPDATE ABONENT1
    SET VAL1 = (SELECT 1
                FROM ABONENT2 A3 WHERE
                A2.FID = A1 .ID )
    WHERE (EXISTS SELECT *
                  FROM ABONENT2 A2
                  WHERE A2.FID =
                  ABONENT1.ID)
```

Important term of applying this recommendation is existence of the index of special field from sub queries. In this case effectiveness is ensured by means of combination of operations with lines set and ID field index. It allows to

not to scan the tables with the biggest amount of records in whole. The verification of results is shown in the table 1, completed by the authors.

Query variation	Time to perform, (ms)	Percent of effectiveness regarding to Listing 1, (%)	Percent of effectiveness regarding Listing 2, (%)
Listing 1 (=)	1750		
Listing 2 (IN)	1328	24.11	
Listing 3 (EXIST)	140	92.46	89.46

Table 1. Comparison of results of queries variations performance

So significant preferring of the variant with EXIST (Listing 5-3) over IN (Listing 5-2) can be explained by the records number proportion in the tables: in A1- more than 100 thousands records, in A2 - 8 records.

Recommendation: if possible, change the comparison for IN or EXIST depending on proportion of the records number in the tables.

6- Absence of the necessary index when performing a *non-correlated* query.

There is an example of query in the (Listing 6-1) performing the non-correlated amendments[4].

(Listing 6-1) – Example of the non-correlated query

```
SELECT A1.ID
FROM ABONENT1 A1
WHERE A1.ID IN (SELECT A2.FID
                FROM ABONENT2 A2 )
```

As it is known, DBMS core does not use the field index if in the section WHERE no conditions is imposed on it. A particular case of breaching this rule is an example of non-related query illustrated in the Listing 4. Sub query is not provided by the section WHERE, nevertheless in this case DBMS core will try to use the index for the field A2.FID in case if it is required. The developer's purpose is to set the index in the field A2.FID. The effectiveness of this recommendation is 47.17%. We can say that the effectiveness of this recommendation will depend on the speed of index scanning of the table A., unlike its full scanning.

7- Unintended prohibition of indexes.

There are various ways of unintended blocking of indexes provided for DMBS MS Firebird SQL, Oracle, part of them is shown in the table 2, completed by the authors.

Search terms leading to blocking of index search	Search terms not blocking the search index	Effectiveness 2 regarding to 1 in Firebird SQL
1	2	3
ID = '992000000'	ID = 992000000	(15-0)/15-100%
E <> 3	E <3 OR E> 3	(31-15)/31=51.61%
E NOT IN ('2', '32')	E<2 OR	(33-15)/33=54.55%
P * 2 <1000	P < 500	(500-16)/500-96.80%

Table 2 .Examples of search terms

Proper applying of a search argument (terms in section WHERE) is critical for Firebird SQL DBMS, but it is not relevant for MS SQL Server in which the optimizer will choose the index (if it is required) in accordance with its price. If the price of non-clustered index is less than the price of clustered index, then, even upon the improper use of search argument, a query will be processed for tolerable period, because the search will be performed by the non-clustered index, in other words, it is impossible to block index search in DBMS MS SQL Server when the clustered index is required.

Recommendation: reformulate the terms blocking the index search, if the DBMS MS Firebird SQL Oracle is used, but it is not important for DBMS MS SQL Server.

8- Rule 10,15, 20% is neglected.

During the experiments, it was found out that using of indexes in queries is appropriate if the query extracts less than 15 % (10, 20) rows from the table. In all other cases, full review of the table will be operating more rapidly. This rule is appropriate for both MS SQL Server and Firebird SQL.

4- Conclusion

We can make the following summary:

1. There are recommendations from the ones presented above, which do not bring tangible gains in time, nevertheless, they are highly recommended for use. These include, for example, a recommendation using synonyms, recommendations for an approximate calculation of the number of records from each table participating in the query.

2. To apply some recommendations, you should pay attention to the structure of the query, in particular - whether sub queries are used, whether operation operators on sets of rows of the type EXIST, IN, UNION and others are used. In case of detecting such properties, it is necessary to evaluate the efficiency of processing each of the sets, possibly by applying on the sets operations such as packing, subtraction, addition, etc., or at least to have an idea of the number of records that satisfy the terms of the query.

3. Some queries can be a special case, for example, the so-called "non-related" queries, and require competent indexing of the fields of tables participating in the query.

4. It is necessary to use the operator DISTINCT accurately because its applying is not always profitable due to dependence on the position of key row in the table.

5. DBMS MS SQL Server and Firebird SQL have shown the different approaches for organization of indexes in their systems.

5- The Results:

1.Check the amount of pseudonyms. If there is more than one table in query, it is better to use pseudonyms.

2.Check the search terms. If the DBMS is not the MS SQL Server, and using of indexes is required, the non-blocking search terms must be used.

3.Check the presence of the clause DISTINCT, if it is present, analyze the distance between the table first row and the key record. If it is located in the second 1/3 part of the table and farther, the query construction must be changed, for instance, use the "IN" or EXIST operators.

4.If there is a few sources used in the clause FROM, estimate, if possible, the number of records returning from each of them and set the terms by decreasing (the number of records complied for the terms must be decreasing).

5.Check the number of the clauses IN or EXIST and analyze the number of the records complied for selection terms.

6.Check availability of relational algebraic operators UNION, INTERSECT, etc., if they exist, then analyze if it is possible to work without these operators. In the process of designing a query, you can try to use only the WHERE clause and a sub query in it, or use relational algebra operators together with sub queries. Choose the best option.

7.Adjust the database service to ensure regular performance of indexes and statistics refreshing.

CONFLICT OF INTERESTS
There are no conflicts of interest.

References

[1] Loshmanov, A.U. Organization of work to maintain the information system of the university / A.U. Loshmanov, Ya.U. Grigoryev, A.N. Petrova // Internet journal; "Science", No 4 (17) [Electronic source]-M.: Science, 2013. – Regime of the access: http://naukovedenie.ru/PDF/66tvn413.pdf, free. – From the screen. - Languages. Russian., English. 2013.

[2] Evseev, G.S. Investigation of the influence of the level of normalization of database tables on the performance time of queries / G.S. Evseev. D.M. Ilyinskaya // Collection of the Conference "Science session EUAP". Saint Petersburg: GUAP, p. 193- 196. 2015.

[3] Miheevich, V. Causes of ineffectiveness of the SQL-queries in Oracle. Optimization of the SQL-queries productivity/ V. Miheevich / System administrator. No 6. p. 47-51. 2015.

[4] Borri, H. Firebird: Manual for databases developers/ H. Borri. – Saint Petersburg.: BHV-Petersburg, 1104 p. 2006.

[5] Kuznetsov S.D. Estimation of efficiency of minimization of restrictions of queries to a DBMS / S.D. Kuznetsov, N.A. Menkovich // Proceedings of the Institute of System Programming RAN No 25 p. 113-130.

[6] Date, K.J. Introduction to database systems: Per. from English. / K.J. Date. - M.: Publishing house "Williams", 1072 p. 2002.

[7] Markin, A.B. Query Building and SQL Programming / A.B. Markin. - M.: Dialogue-MIFI, 318 p. 2008.

[8] Bludov, I.V. Features of the SQL table expressions and their correspondence with the concepts of the relational data model / I. V. Bludov. // Proceedings of the Institute of System Programming RAN. No 24. P. 417-436. 2013.

[9] Atroschenko, B.A. Databases / B.A. Atroschenko, R.A. Dyachenko, N.D. Chiglikova, B.E. Belchenko, E.S. Beloded, I.S. Loba. - Armavir Armavir State Pedagogical Academy, 80 p. 2015.

[10] Malkov, O.B Work with Transact-SQL / O. B. Malkov, M.B. Devyaterikova. - Omsk: Federal State Budget Educational Institution of Higher Professional Education “Omsk State Technical University”, 136p. 2015.

التحقيق في توصيات استكشاف الأخطاء وإصلاحها من عدم فعالية الاستعلامات لجداول قاعدة البيانات

الخلاصة

تتمثل إحدى الطرق لزيادة إنتاج العمل للتطبيقات المرتبطة بقاعدة البيانات في تحسين استعلامات جداول قاعدة البيانات. تزداد أهمية مشكلة تحسين الاستعلام مع نمو عدد السجلات إلى عقود ومئات الآلاف من الجداول التي يشير إليها الاستعلام. هناك العديد من التوصيات حول استكشاف أخطاء تباطؤ أداء الاستعلامات. ومع ذلك، لا توجد نتائج تحقيقات تجريبية حول فعالية تطبيق التوصيات الأخرى المقدمة في الأدبيات العلمية. وقد أجرى المؤلفون تجارب عددية للتحقيق في التوصيات، سواء الموجودة أو المقترحة من قبل المؤلفين، لتحديد القيم الكمية من فعالية تطبيق التوصيات. يتم توضيح عدد من التوصيات من خلال أمثلة لتطبيق التوصيات بما في ذلك العديد من التغييرات البديلة. يصف العمل الاختلافات في فعالية تطبيق التوصيات في أنظمة مختلفة من إدارة قواعد البيانات الواردة في هذه المقالة. يصف العمل القيم النسبية لفعالية تطبيق التوصيات مقارنة مع استعلامات المصدر. نتيجة للمادة، طور المؤلفون خوارزمية تحليل نص الاستعلام وتطبيق التوصيات. نتائج التحقيق توسيع المعرفة على التطبيق العملي لتطبيق كل من التوصيات المعروفة والجديدة لتحسين الاستعلامات لجداول قاعدة البيانات. تحتوي هذه المقالة على توجه عملي وهي مفيدة بشكل منهجي للمبرمجين للمرة الأولى في مجال لغة الاستعلامات المنظمة.

الكلمات الدالة: DBMS، تعليمات SQL. تحسين التطبيقات؛ قاعدة البيانات، أوراكل.