Increase the effectiveness of search databases queries using the algorithm of bitmap scales

T N Nosova, O B Kalugina* and I I Barankova
Nosov Magnitogorsk State Technical University, 38 Lenina avenue, Magnitogorsk, 455000, Russian Federation
*Corresponding autor: kalugina.olga@bk.ru

Abstract. Query optimization is an important part of any database application. The most effective method of an acceleration of search queries is use of index structures. It is known that queries which use the indexes (clustered and not clustered) are effective in the case of low percent of the repeating values in columns. If indexable data are not selective, use of the majority of types of indexes is not effective. Main objective of the performed work is extending of SQL Server MS for creation of indexes and increase in productivity of search queries. In the furtherance of this goal the high level language Net-application using C# is created. Embedded algorithm creates a bit scales for processing of relational tables columns with a large count of the duplicated values. In article it was conducted the review of the main existing methods of search query efficiency increasing and types of the index structures used in different database management systems. Examples of action of application for selection of values with use of bit indexes are shown. Testing of the created software product on tables with different cardinality allows to draw conclusions about the considerable abbreviation of time of data handling in case of using of bit indexes in comparison with other search algorithms.

Keywords: SQL query optimization, data indexing, binary indexes, data dictionary.

1. Introduction

The optimization of queries is an important part of the work of any database application. The most effective method of accelerating search queries is the use of different types index structures. It is known that queries using clustered and nonclustered indexes are as productive as possible when relational table columns contain a small percentage of reduplicate values. If the data being processed is not selective, the use of most types of indexes is inefficient.

The main goal of the research is the expansion of the MS SQL Server environment capabilities for indexing data and increasing the performance of search queries. To achieve this goal, a Net application was created in the C#, using the algorithm for generating bitmaps for processing of relational tables columns with a large number of duplicated values.

The article reviews the main existing methods to increase the efficiency of query, as well as the types of index structures used in various database management systems. Examples of the application's work on sampling values using bit indexes are demonstrated.

Testing the created software on tables with different cardinality allowed us to draw conclusions about a significant reduction in the processing time when applying bitmap scales compared to other search algorithms.

An effective way to data access processing is the critical task practically in all modern web applications. As a result, projects interacting with databases should pay attention to query optimization, otherwise the system response duration to user query becomes unacceptable.

2. Basic approaches to optimizing SQL queries
A number of articles and reviews have been devoted to the optimization of queries to relational databases. [1-8, 14]. To date, a fairly large number of methods have been proposed to increase the query efficiency. The most common of them are: the study of the query execution plan, the indexing of the fields of relational tables (RT), and the analysis of the degree of indices selectivity.

The query plan is created in the data processing optimization phase by the database engine component, called the query optimizer. The latter, taking into account many different factors, tries to find the most efficient algorithm for data processing [9].

The use of indexing mechanisms is one of the main ways to reduce the execution time of queries. Valid indexes can significantly reduce the data processing time. Clustered and nonclustered indexes help the database server to find the result much faster, using different versions of balanced B-trees and hash tables [10].

Many sources recommend the rule for the optimal systems creation to regulate the setting of indexes on foreign keys of table relationships. A system just searches for a record in different tables exactly by indexed fields [4].

In most cases it is recommended:
- To update statistics for relational tables.
- To simplify the SELECT command.
- To create indexes on columns that are often used in the WHERE clause of built-in SQL statements or end users queries.
- To index columns often used in SQL statements to join tables, using non-unique indexes on foreign key columns.
- To use for indexing only columns containing a small percentage of rows with the same value [2-6].

The issue of optimizing queries to relational databases discussed in many articles and reviews [1-8]. To date, a large number of methods have been proposed that increase the efficiency of query execution. The most common ones are: study of the query execution plan, indexing the fields of relational tables (RT) and analysis of the degree of indices selectivity.

The query plan is created in the data processing optimization phase by the database engine component, called the query optimizer. The latter, taking into account many different factors, tries to find the most efficient algorithm for data processing [9].

The use of indexing mechanisms is one of the main ways to reduce the execution time of queries. Correctly constructed indexes can significantly reduce the processing time of data. Clustered and nonclustered indexes help the database server to find results much faster, using different versions of balanced B-trees and hash tables for this purpose [10].

It should be taken into account that the indexes slow down the execution of DML (data manipulation language) commands, and their maintenance requires time and increases the cost of processing. Many DBMSs block the use of indexes in cases of: the index field is used in calculated expressions, as a comparison operand with non-indexed field values, in operations that use a comparison with an undefined NULL value, or is a parameter of built-in or user-defined functions [4].

DBMS developers have their own approaches to the organization of indexes. The currently known DBMS uses different kinds of indexes (Table 1)

| Index type            | MySQL       | PostgreSQL SQL | MS SQL | Oracle |
|-----------------------|-------------|----------------|--------|--------|
| B-Tree index          | Yes         | Yes            | Yes    | Yes    |
| Supported spatial indexes | B-Tree index | Yes            | Yes    | Yes    |
| Hash index            | only in Memory type tables | Yes | No    | No    |
| **Bitmap index**      | No          | Yes            | **No** | Yes    |
| Reverse index         | No          | No             | No     | Yes    |
| Inverted index        | Yes         | Yes            | Yes    | Yes    |
Microsoft SQL Server works with the following types of indexes: hash, nonclustered indexes with memory optimization, clustered, nonclustered, unique, columnstore, index with columns enabled, index on computed columns, filtered, spatial, xml, fulltext. The data is stored in the table with a sorted form only if a clustered index is created for this table.

In most cases, indexing of the fields, used in search condition, improves query performance. However, in reality, this is determined by the parameters of the selectivity of the index and the coefficient of repetitive values. The coefficient of repeated values (the density of distribution of values) is information about the number of duplicates in the analyzed column or a combination of columns. It is calculated as:

\[
\frac{1}{\text{number of distinct values}}.
\]

If the distribution density exceeds 10%, then the index can be considered useless. Selectivity (selectivity) of the index is an indicator of how many lines of the total number are accounted for by one key index value. This is the estimated number of rows that can be selected when searching for each index value.

Index selectivity \( S \) was calculated as:

\[
S = \frac{n}{\text{number of rows in the table}},
\]

where \( n \) is the number of different instances of the index value in the table. A unique index has the highest possible selectivity of 1, while an index for a column with a boolean data type, for example, has practically the lowest selectivity.

The optimizers of most DBMS find the coefficient for calculating the selectivity for the first access to the table, and store it in memory for use in calculating plans for subsequent queries to this table [11]. The most useful for the optimizer are the query criteria for indexed fields with high selectivity (low density), because they can reliably predict Input / Output operations count, required when running a query [12].

In general, the more duplicates in the indexed column, the worse the index works. Therefore, all the above recommendations to increase query performance are not suitable for searching data on fields with low selectivity.

3. Task description

The main goal of our work is the extension of the MS SQL Server environment capabilities for data indexing and increasing the performance of search queries. To achieve this goal, a software tool has been developed. It processes a queries to relational tables containing fields with a large fraction of duplicate values and uses bit indexes for this.

Little unique values traditionally contain such fields of tables as: gender, position of employee, region and city of location, category of goods or employee, academic semester, assessment, place of publication of literature, etc. But such columns are often used in simple or compound predicates for the selection of SQL queries.

4. Proposed solution

To increase of data searching in the RT for fields with low selectivity, an application was developed for the Net platform in the programming language C#. Application implements an efficient search query mechanism using bit indexes.

Usually in the B*-tree there is a one-to-one correspondence between the record of an index and the row of a table. A bitmap is used to refer to a large number of table rows at a time. Such indexes are suitable for data with a small number of different values, subject to an infrequent change.

The bitmap indexes were investigated as a means of the efficiency increasing of queries in the MS SQL Server database.

The method of bitmap indexes was made as creation of individual bitmaps (sequences 0 and 1) for each possible column value, where each bit corresponds to a string with an indexed value, and its value equal to 1 means that the record corresponding to the bit position contains an index value for this column or properties [10].

The program was tested on the example of the relational table "Employees", the structure of which...
is described in the following SQL instruction.

```
CREATE TABLE Employees
(Id_Empl AS Id INTEGER PRIMARY KEY,
LastName AS Full_name VARCHAR(30) NOT NULL,
Sex AS Пол VARCHAR(10) CHECK (Sex='м' OR 'ж'),
Position AS Position VARCHAR(25) CHECK
IN('Programmer', 'designer', 'administrator'),
Group_Experience AS Experience_Group
VARCHAR(10) CHECK IN ('I', 'II', 'III')
);
```

The table can be filled with data from one of several connected files, which allows us to vary the number of the table records and change the values of the attributes.

| ID  | Name       | Position | Experience_Group |
|-----|------------|----------|------------------|
| 101 | Pobens     | programmer | I                |
| 102 | Ivanov     | designer   | II               |
| 103 | Vogerler   | programmer | I                |
| 104 | Pizzir     | administrator | III            |
| 105 | Fourth     | designer   | I                |
| 106 | Sokolov    | programmer | I                |
| 107 | Simon      | administrator | I            |
| 108 | Chen       | designer   | I                |
| 109 | Anclos     | programmer | II               |
| 111 | Molchov    | administrator | I            |

**Figure 1.** Test example of a table with ten records.

The above SQL statement demonstrates that only the **Id** column has unique values and that a clustered index will be built for it by Microsoft SQL Server.

To increase the speed of query execution by other table fields, the general recommendation is to create a nonclustered index one by one or a collection of columns. In fact, this approach does not lead to a reduction in the execution time of the query, since the index is selective by fields: position, gender, trainee group, position is low. In the test case, the "Position" field contains only three unique values, "Sex" – two unique values.

**Table 2.** The density of the distribution of values in the fields of the table "Employees".

| Attribute name           | Density of data distribution = 1 / number of unique values |
|--------------------------|------------------------------------------------------------|
| Sex                      | 0,5                                                         |
| Position                 | 33,33                                                      |
| Group_Experience         | 33,33                                                      |

As we can see from Table 1, MS SQL Server does not present bitmap-based indexes, therefore, the developed application extends the built-in DBMS tools to solve query optimization problems.

The developed Net-project contains several static methods of the Program class, sequentially implementing the mechanism of bitmaps (maps) for the table in question (Figure 2).
The static `Create_Bit_Map()` method is used to create a bitmap for a given table field (see Figure 3). Accepts as parameters: a reference to an array of employees, a string array, found in a column of unique values, and the column number for which a bitmap is created.

```csharp
static Dictionary<string, int[]> Create_Bit_Map(string[,] arr_Emp, string[] arr_Keys, int number_field)
{
    int N = arr_Emp.GetLength(0);  //The size of a bit scale
    var bitmap = new Dictionary<string, int[]>();
    // Adding items to the dictionary
    for (int i = 0; i < arr_Keys.Length; i++)
    {
        //Bit-scale array for the current key
        var temp_arr = new int[N];
        for (int j = 0; j < N; j++)
        {
            if (arr_Emp[i, number_field] == arr_Keys[i])
                temp_arr[j] = 1;
            else temp_arr[j] = 0;
        }
        //Forming a bitmap and adding it to the dictionary
        bitmap.Add(arr_Keys[i], temp_arr);
    }
    //Bitmap printing
    for (int j = 0; j < N; j++)
    {
        Console.Write(bitmap[arr_Keys[1]][j]);
    }
    Console.WriteLine();

    return bitmap;
}
```

Figure 3. Implementing the algorithm for composing the bitmap for a given table field.
The method returns the finished bitmap-index of the specified table field. It programmed using the built-in class “Dictionary”. Dictionary is an analog of associative arrays in other programming languages working with {Key-> Value} pairs [13].

In our example, it is a set of [string, int []], where Key is the next unique value of the field, represented as a string; the Value is an integer one-dimensional array containing 0 and 1. If value[i]=1 then it means the coincidence of the value of the field with the next unique value in the current record. Figure 4 shows the result of the program for a test table of 10 rows.

| ID | Name         | Position | Experience_Group |
|----|--------------|----------|------------------|
| 101| Pobens       | programmer | 1                |
| 102| Ivanov       | designer  | 1                |
| 103| Vogeler      | programmer | 1                |
| 104| Pizir        | administrator | 11               |
| 105| Fourth       | designer  | 1                |
| 106| Sokolov      | programmer | 1                |
| 107| Simon        | administrator | 1                |
| 108| Chen         | designer  | 1                |
| 109| Ancles       | programmer | 1                |
| 110| Moskow       | administrator | 1                |

Figure 4. The result of the bitmaps creation for the two fields of the relational table.

The bitwise multiplication method public static void Bit_Devision(), is designed to implement a query with compound selection criteria for fields with a small percentage of unique values. It takes as parameters: a table reference, references to one or more bit indexes, and search criteria represented as text strings.

```csharp
public static void Devision_BitMaps(string[ , ] , Dictionary <string, int[]> , string , Dictionary <string, int[]>, string)
{/…}
```

5. Results

The subprogram implements the bitwise multiplication algorithm with a several bit scales and it prints records, accord with search condition (Figure 5)

```
Figure 5. The query of two search criteria result
```

To support of queries with a complex search condition, several overloaded methods were created. The developed search methods function by two and by three given criteria. Overloading of method is used to realize the possibility of using subprograms of the same name, with differ set of parameters.

The tests of the created application on tables of different sizes showed a significant reduction in query execution time when using bitmap-indexes in comparison with to line-by-line search. To confirm this thesis, the program has been added code that performs the search query by full scanning of the table.
Figure 6. Comparison of the speed of query execution using bit scales and full table scan.

The results of the program for a table of 100 and 1000 records are given in Table 3.

Table 3. Comparison of the search query execution time for different approaches.

| Search algorithm    | Records count |
|---------------------|---------------|
|                     | 100 | 1000 |
| full table scan     | 5 mc | 23 mc |
| Bitmap indexes      | 2 mc | 12 mc |

It should be noted that some resources are spent on compiling and storing the bitmap, so the binary index is most effective in tables with infrequent change of values.

6. Conclusions
1. The result of the developed software using is an increase in the speed of search when processing data in relational tables.
2. The developed software application extends the built-in capabilities of the MS SQL Server database for indexing data and optimizing search queries.
   The main types of indexes are quite effective in respect of relational tables fields with unique values, or with a low value density.
   The binary indexes using is justified for fields with a large proportion of duplicate values, in the case of availability of disk space resources and when the modification of the tables is not intensive.
   The developed software, using the algorithm of bitmap indexes, make it possible to improve the performance of search queries in relational tables for fields with a small share of unique values.

References

[1] Stain D A and Hasovskih V P 2014 Methods of optimization and increase in access efficiency to data in management information systems the organization Fundatental researches 12-10 2114-19
[2] Dalvi Nilesh and Suciu Dan 2007 Efficient query evaluation on probabilistic databases The VLDB Journal 16 pp 523-544
[3] Patel Dhaval and Patel Pratik 2015 An Approach for Query Optimization by using Schema Object Base View Int J Comput Appl T 119 pp 21-25
[4] Phrolov K M and Knjazev V N 2015 Query tuning to databases on the basis of the index mechanism Collection scientific articles XII int. sci.technic. conf. "New Information Technologies and Systems" pp 201-204
[5] Indrayana I N E, Wirasyanti N M and Sudiartha I KG 2018 Heuristic query optimization for query multiple table and multiple clause on mobile finance application J. Phys.: Conf. Ser. 953 012049
[6] Petuhov I S 2017 Algorithm of determination of necessary indexes for query tuning with connection of two tables in MYSQL (INNODB) DBMS Scientific bulletin of the state scientific research institute of civil aviation (GosNII GA) 16 98-107
[7] Riggs S and Krossing X 2013 PostgreSQL 9 Administration Cookbook (Mumbai: BirminghamPackt Publ) 368
[8] Tarasov C V 2015 The DBMS for the programmer. Databases from within (Moscow: Solomon-Press Publ) 320
[9] Tumanov VE 2016 *Design of data stores for applications of systems of business awareness* (Moscow: Intuit Publ) 958

[10] Petkovic Dusan 2013 *Microsoft SQL Server 2012 A Beginners Guide* (St. Petersburg: BHV-Petersburg) 816

[11] Barrie X 2004 *The Firebird Book: A Reference for Database Developers* (New York: Apress Publ) 1128

[12] Henderson K 2003 *The Guru's Guide to SQL Server Architecture and Internals* Addison – Wesley Publ 685

[13] Skit J 2014 *C# for professionals: programming subtleties* (Moscow: Williams Publ) 610

[14] Nosova T N, Bykova T V, Bulatov R R and Mikhailova U V 2015 Protection of ORACLE databases *Actual problems of modern science, technology and education* (Magnitogorsk: NMSTU Publ) 188-191