ИНФОРМАЦИОННОЕ, МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ТЕХНИЧЕСКИХ СИСТЕМ

УДК 004.421 https://doi.org/10.18503/2311-8318-2018-1(38)-63-67

Носова Т.Н., Калугина О.Б.

Магнитогорский государственный технический университет им. Г.И. Носова

Использование алгоритма битовых шкал для увеличения эффективности поисковых запросов, обрабатывающих данные с низкой избирательностью

Оптимизация запросов является важной частью работы любого приложения, взаимодействующего с базами данных. Наиболее эффективным методом ускорения поисковых запросов является применение индексных структур разного вида. Известно, что запросы, использующие кластеризованные и некластеризованные индексы, максимально производительны в том случае, когда столбцы реляционных таблиц содержат невысокий процент повторяющихся значений. Если обрабатываемые данные не избирательны, использование большинства видов индексов является неэффективным. Основной целью представленной работы является расширение возможностей технологий в среде MS SQL Server по индексированию данных и увеличению производительности поисковых запросов. Для реализации этой цели было создано Net-приложение на языке программирования высокого уровня C#, использующее алгоритм формирования битовых шкал для обработки столбцов реляционных таблиц с большим количеством дублированных значений. В статье сделан обзор основных существующих методов, повышающих эффективность выполнения запросов, а также видов индексных структур, используемых в различных системах управления базами данных. Продемонстрированы примеры работы приложения по выборке значений с использованием битовых индексов. Тестирование созданного программного продукта на таблицах с разной кардинальностью позволило сделать выводы о значительном сокращении времени обработки данных при применении битовых шкал по сравнению с другими алгоритмами поиска.

Ключевые слова: оптимизация запросов SQL, индексирование данных, двоичные индексы, словарь данных.

Введение

Практически во всех современных веб-приложениях эффективные способы доступа к обработке данных являются критически важными задачами. Как следствие, в проектах, взаимодействующих с базами данных, следует уделять внимание оптимизации запросов, иначе время отклика системы на запросы пользователей становится неприемлемым.

Основные подходы к оптимизации SQL-запросов

Вопросу оптимизации запросов к реляционным базам данных посвящено множество статей и обзоров [1-8]. На сегодняшний момент предложено достаточно большое количество методов, повышающих эффективность выполнения запросов. К самым распространенным из них относятся: изучение плана выполнения запроса, индексирование полей реляционных таблиц (РТ) и анализ степени избирательности индексов.

План запроса создается в фазе оптимизации обработки данных компонентом ядра базы данных, называемым оптимизатором запросов. Последний, принимая во внимание множество различных факторов, пытается подобрать наиболее эффективный алгоритм обработки данных [9].

Применение механизмов индексирования является ещё одним из основных способов сокращения времени выполнения запросов. Правильно построенные индексы могут значительно сократить время обработки данных. Кластеризованные и некластеризованные индексы помогают серверу баз данных находить результат значительно быстрее, используя для этого разные варианты сбалансированных B-деревьев и хеш-таблиц [10].

Многие источники рекомендуют правило формирования оптимальных систем, регламентирующее выставление индексов на внешних ключах табличных связей. Именно по этим полям система осуществляет поиск той или иной записи в разных таблицах [3].

Также следует проводить анализ запросов, используемых в подсистеме и индексировать именно те поля, которые используются для сортировки или поиска.

В большинстве случаев рекомендуется:

- Обновлять статистику для реляционных таблиц.
- Упрощать команду SELECT.
- Создавать индексы на столбцах, которые часто используются в разделе WHERE встроенных операторов SQL или запросов, используемых конечными пользователями.
- Индексировать столбцы, часто используемые в операторах SQL для соединения таблиц, применяя для этого неуникальные индексы на столбцах внешних ключей.
- Использовать для индексирования только те столбцы, в которые входит небольшой процент строк с одним и тем же значением [2-6].

Следует учитывать, что индексы замедляют выполнение команд DML (языкманипулированияданными), а их сопровождение требует времени и увеличивает стоимость обработки. Многие СУБД блокируют использование индексов, если: индексное поле используется в вычисляемых выражениях, в качестве операнда сравнения со значениями неиндексированного поля, в операциях, использующих сравнение с неопределенным значением NULL, или является параметром встроенных или пользовательских функций [4].

Необходимо отметить, что в вопросе организации индексов у разработчиков различных СУБД имеются свои подходы.

Известные на сегодняшний момент СУБД используют разные виды индексов **(табл. 1)**.

Таблица 1

Сводная таблица типов индексов в различных СУБД

Тип индекса	MySQL	Postgre SQL	MS SQL	Oracle
B-Tree index	Есть	Есть	Есть	Есть
Поддерживаемые пространственные индексы (Spatial indexes)	B-Tree index	Есть	Есть	Есть
Hash index	Только в таблицах типа Memory	Есть	Нет	Нет
Bitmap index	Нет	Есть	**Нет**	Есть
Reverse index	Нет	Нет	Нет	Есть
Inverted index	Есть	Есть	Есть	Есть
Partial index	Нет	Есть	Есть	Нет
Function based index	Нет	Есть	Есть	Есть

СУБД Microsoft SQL Server работает со следующими видами индексов: хэш, некластеризованные индексы с оптимизацией для памяти, кластеризованный, некластеризованный, уникальный, columnstore, индекс с включенными столбцами, индекс на вычисляемых столбцах, фильтруемый, пространственный, xml, полнотекстовый [11]. Данные в таблице хранятся в отсортированном виде только в случае, если создан кластеризованный индекс для этой таблицы.

В большинстве случаев индексирование полей, используемых в условиях отбора, повышает производительность запросов. Однако в действительности это определяется параметрами избирательности индекса и коэффициента повторяющихся значений.

Коэффициент повторяющихся значений (плотность распределения значений) – это информация о числе дубликатов в анализируемом столбце или комбинации столбцов. Он вычисляется как:

1/(число различающихся значений).

Если плотность распределения значений превышает 10%, то индекс можно считать бесполезным.

Избирательность (селективность) индекса – это показатель того, сколько строк от общего числа приходится на одно ключевое значение индекса. Это оценочное количество строк, которые могут быть выбраны при поиске по каждому значению индекса.

Селективность (selectivity) индекса (Индексная селективность, S) вычислялась как:

S = n / количество строк в таблице,

где n – количество различных экземпляров значения индекса в таблице. Уникальный индекс имеет максимально возможную селективность, равную 1, в то время как индекс, например для столбца с типом данных BOOLEAN, имеет практически самую низкую селективность.

Оптимизаторы большинства СУБД отыскивает коэффициент для вычисления селективности при первом обращении к таблице и сохраняет его в памяти для использования при вычислении планов при последующих запросах к этой таблице [12].

Наиболее полезными для оптимизатора являются критерии запроса по индексированным полям с высокой избирательностью (соответственно низкой плотностью), т.к. позволяют надежно предсказать, какое количество операций ввода-вывода потребуется при выполнении запроса [13].

В общем случае, чем больше дубликатов в индексируемом столбце, тем хуже работает индекс. Поэтому все вышеперечисленные рекомендации по увеличению производительности запросов не подходят для поиска данных по полям с низкой избирательностью.

Цель работы

Основной целью представленной работы является расширение возможностей технологий в среде MS SQL Server по индексированию данных и увеличению производительности поисковых запросов.

Для реализации этой цели было разработано программное средство, обрабатывающее запросы к реляционным таблицам, содержащим поля с большой долей повторяющихся значений, и использующее для этого битовые индексы.

Малоуникальные значения традиционно содержат такие поля таблиц, как: пол, должность сотрудника, область и город местонахождения, категории товара или сотрудника, учебный семестр, оценка, место издания литературы и т.п. Но такие столбцы часто используются в простых или составных предикатах отбора SQL-запросов.

Для решения проблемы поиска данных в РТ по полям с низкой избирательностью было разработано приложение для платформы Net на языке программирования C#, реализующее эффективный механизм поисковых запросов, использующий битовые индексы.

Обычно в B*-дереве имеется однозначное соответствие между записью индекса и строкой таблицы. Битовая карта используется для ссылки на большое количество строк таблицы одновременно. Такие индексы подходят для данных с небольшим количеством различных значений, подвергающихся нечастому изменению.

Объектом исследования являлись двоичные (bitmap)-индексы как средство повышения эффективности обработки поисковых запросов в СУБД MS SQL Server.

Метод битовых индексов заключается в создании отдельных битовых карт (последовательностей 0 и 1) для каждого возможного значения столбца, где каждому биту соответствует строка с индексируемым значением, а его значение равное 1 означает, что запись, соответствующая позиции бита, содержит индексируемое значение для данного столбца или свойства [14] .

Программа тестировалась на примере реляционной таблицы «Сотрудники», описать структуру которой позволяет следующая инструкция SQL.

```
CREATE TABLE Employees
(Id_Empl AS Id INTEGER PRIMARY KEY,
LastName AS ФИО VARCHAR(30) NOT NULL,
Sex AS Пол VARCHAR(10) CHECK (Sex='м' OR 'ж'),
Position AS Должность VARCHAR(25) CHECK
IN('программист', 'дизайнер', 'администратор'),
Group_Experience AS Стаж_группа
VARCHAR(10) CHECK IN ('I', 'II', 'III')
);
```

Заполнение таблицы данными может производиться из одного из нескольких подключаемых к проекту файлов, что позволяет варьировать количество записей в таблице и изменять значения атрибутов **(рис. 1)**.

Приведенная выше инструкция SQL демонстрирует, что только столбец Id имеет уникальные значения и для него системой Microsoft SQL Server будет построен кластеризованный индекс.

Для увеличения скорости выполнения запросов по другим полям таблицы общей рекомендацией является создание некластеризованного индекса по одному или совокупности столбцов. В действительности такой подход к сокращению времени выполнения запроса не приводит, так как избирательность индекса по полям: должность, пол, стажевая группа, должность – низкая. В тестовом примере поле «Должность» содержит всего три уникальных значения, «Пол» – два.

Как следует из **табл. 1**, в MS SQL Server не представлены индексы на основе битовых карт, следовательно, разработанное приложение расширяет встроенные средства СУБД по решению задач оптимизации запросов.

Разработанный Net-проект содержит несколько статических методов класса Program, последовательно реализующих механизм битовых шкал (карт) для рассматриваемой таблицы **(рис. 2)**.

Статический метод Ctreate_Bit_Map( ) предназначен для создания битовой шкалы заданного поля таблицы **(рис. 3)**. Принимает в качестве параметров ссылку на массив сотрудников, строковый массив встречающихся в столбце уникальных значений и номер столбца, для которого создается битовая карта.

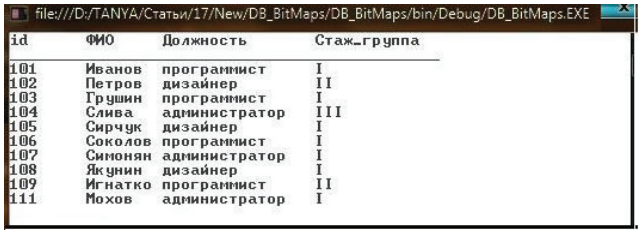

Рис. 1. Тестовый пример заполнения таблицы десятью записями

Таблица 2

Плотность распределения значений в полях таблицы «Employees»

Название поля таблицы	Плотность распределения данных = 1/число уникальных значений
Sex	0,5
Position	33,33
Group_Experience	33,33

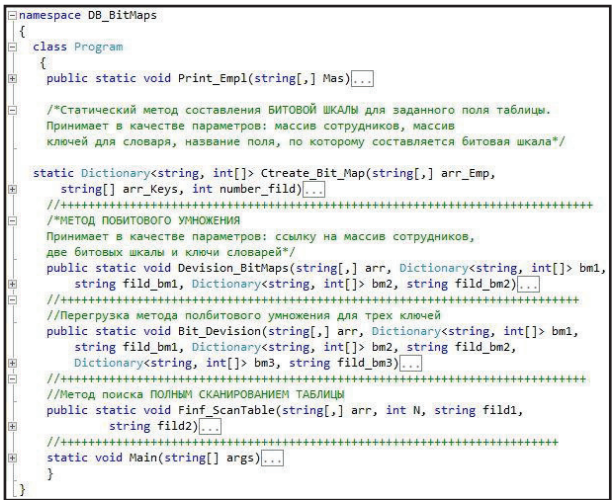

Рис. 2. Структура проекта DB_BitMaps

```
static Dictionary<string, int[]> Ctreate_Bit_Map(string[,] arr_Emp,
    string[] arr_Keys, int number_fild)
   {
   int N = arr_Emp.GetLength(0); //размер битовой шкалы
    var bitMap = new Dictionary<string, int[]>();
     // Добавление элементов в словарь
     for (int i = 0; i < arr_Keys.Length; i++)
       {//массив битовой шкалы для очередного ключа
         var temp_arr = new int[N];
           for (int j = 0; j < N; j++)
             {
                  if (arr_Emp[j, number_fild] == arr_Keys[i])

                      temp_arr[j] = 1;
                  else temp_arr[j] = 0;
            }
            //Формирование битовой шкалы и добавление ее в словарь
            bitMap.Add(arr_Keys[i], temp_arr);

            //Печать битовой шкалы
            for (int j = 0; j < N; j++)
            { Console.Write(bitMap[arr_Keys[i]][j]); }
            Console.WriteLine();
         }
         return bitMap;
      }
```

Рис. 3. Реализация алгоритма составления битовой шкалы для заданного поля таблицы

Метод возвращает готовый bitmap-индекс заданного поля таблицы, программно реализованный с использованием встроенного класса Dictionary – словаря данных.

Dictionary – аналог ассоциативных массивов в других языках программирования, работающих с парами{Key->Value}[15].

В нашем случае это набор {string, int[ ]}, где ключом Key будет являться очередное уникальное значение поля, представленное в виде строки; в качестве Value используется целочисленный одномерный массив, содержащий 0 и 1. Единица на очередной позиции массива соответствует совпадению у очередной записи значения поля с очередным уникальным значением.

Для тестового примера для таблицы размером в 10 строк результат работы программы следующий **(рис. 4)**:

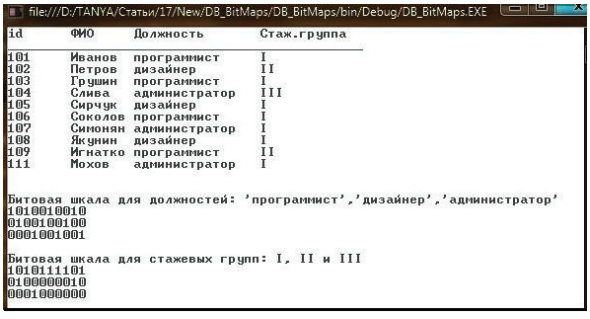

Рис. 4. Результат работы программы по формированию битовых шкал для двух полей реляционной таблицы

Следующий метод побитового умножения public static void Bit_Devision() предназначен для реализации запроса с составными критериями отбора по полям с малым процентом уникальных значений.

Принимает в качестве параметров ссылку на таблицу, ссылки на один или несколько битовых индексов и критерии поиска, представленных в виде текстовых строк.

public static void Devision_BitMaps(string[,] , Dictionary <string, int[ ]> , string , Dictionary <string, int[]>, string)

{/…}

Подпрограмма реализует алгоритм побитового умножения для нескольких битовых шкал и выводит на печать те строки таблицы, для которых выполняется заданный набор критериев отбора **(рис. 5)**.

Для поддержки реализации запросов с составными условиями отбора предусмотрено несколько перегружаемых методов для поиска по двум и трем заданным критериям. Процесс перегрузки методов использован для реализации возможности применения одноимённых подпрограмм, различающихся набором передаваемых параметров.

Тестирование созданного приложения на таблицах разного размера показало существенное сокращение времени выполнения запроса при использовании bitmap-индексов по сравнению с построчным просмотром таблицы. Для подтверждения этого тезиса в программу добавлен программный код, осуществляющий поисковый запрос полным сканированием таблицы.

Результаты работы программы для таблицы размером 100 и 1000 записей приведены в **табл. 3**.

Следует отметить, что некоторые ресурсы тратятся на составление и хранение самой битовой карты, поэтому двоичный индекс максимально эффективен в таблицах с нечастым изменением значений.

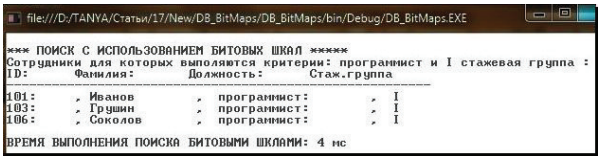

Рис. 5. Результат выполнения запроса с двумя критериями отбора

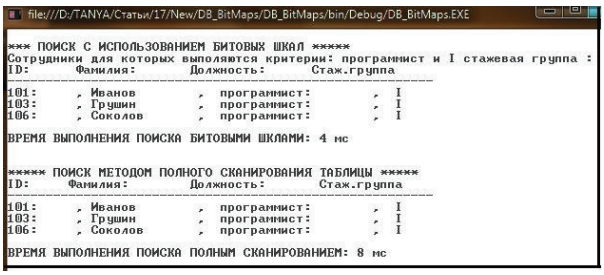

Рис. 6. Сравнение скорости выполнения запроса при использовании битовых шкал и полного сканирования таблицы

Таблица 3

Сравнение времени (мс) выполнения поискового запроса с использованием разных подходов

Использованный алгоритм поиска	Количество записей таблицы	
	100	1000
Полное сканирование таблицы	5	23
Использование битовых индексов	2	12

Результаты

1. Результатом применения разработанного программного продукта является повышение быстродействия поиска при обработке данных в реляционных таблицах.

2. Разработанное программное приложение расширяет встроенные возможности СУБД MS SQL Server по индексированию данных и оптимизации поисковых запросов.

Заключение

Основные виды индексов достаточно эффективны применительно к полям реляционных таблиц с уникальными значениями, либо с низкой плотностью значений.

Применение бинарных индексов оправдано для полей с большой долей повторяющихся значений в том случае, если имеется достаточное количество ресурсов дискового пространства и модификация таблиц не является интенсивной.

Разработанное программное средство, использующее алгоритм битовых шкал, позволяет повысить производительности поисковых запросов в реляционных таблицах по полям с малой долей уникальных значений.

Список литературы

1. Стаин Д.А., Часовских В.П. Методы оптимизации и повышения эффективности доступа к данным в информационных системах управления организацией // Фундаментальные исследования. 2014. № 12-10. С. 2114-2119.
2. Оптимизация производительности выполнения запросов (SQL Server Compact) //Microsoft. Developer Network [Электронный ресурс]. Режим доступа: https://msdn.microsoft.com/ru-ru/library/ms172984.aspx. Заглавие с экрана. (Дата обращения: 10.10.2017).
3. Фролов К.М., Князев В.Н. Оптимизация запросов к базам данных на основе механизма индексирования // Сборник научных статей XII международной научно-технической конференции «Новые информационные технологии и системы». 2015. С. 201-204.
4. Типичные причины неоптимальной работы запросов и методы оптимизации // 1С: ИТС. Информационно-технологическое сопровождение пользователей 1С: Предприятие [Электронный ресурс]. Режим доступа: https://its.1c.ru/db/metod8dev#content:5842:hdoc. Заглавие с экрана. (Дата обращения: 10.10.2017).
5. Optimizing Database Structure [Электронный ресурс]. Режим доступа: https:// dev.mysql.com/ doc/refman /5.7/ en/optimizing-database-structure.html. Заглавие с экрана. (Дата обращения: 10.09.2017).
6. Петухов И.С. Алгоритм определения необходимых индексов для оптимизации запросов с соединением двух таблиц в СУБД MySQL (INNODB) // Научный вестник ГосНИИ ГА. 2017. № 16. С. 98-107.
7. Ригс С., Кросинг Х. Администрирование PostgreSQL 9. Книга рецептов / пер. с англ. Самохвалова Е.В. М.: ДМК Пресс, 2013. 368 с.
8. Тарасов С.В. СУБД для программиста. Базы данных изнутри. М.: Солон-Пресс, 2015. 320 с.
9. Туманов В.Е. Проектирование хранилищ данных для приложений систем деловой осведомленности (Business Intelligence Systems). М.: Национальный Открытый Университет «ИНТУИТ», 2016. 958 с.
10. SQL Server 2000 [Электронный ресурс]. Режим доступа: http://www.intuit.ru/studies/courses/68/68/lecture/2016?page=3. Заглавие с экрана. (Дата обращения: 10.10.2017).
11. Microsoft. Developer NetWork [Электронный ресурс]. Режим доступа: https://msdn.microsoft.com/ru-ru/library/ms175049(v=sql.120).aspx. Заглавие с экрана. (Дата обращения: 10.09.2017).
12. Борри Х. Firebird: руководство разработчика баз данных. СПб.: БХВ-Петербург, 2013. 1104 с.

13. Хендерсон К. Профессиональное руководство по SQL Server: структура и реализация. М.: Вильямс, 2006. 1044 с.
14. Академия Microsoft: Распределенные базы и хранилища данных [Электронный ресурс]. Режим доступа: http://www.intuit.ru/studies/courses/1145/214/lecture/5523?page=2 intuit. Заглавие с экрана. (Дата обращения: 10.10.2017).
15. Скит Д. C# для профессионалов: тонкости программирования. М.: Вильямс, 2014. 610 с.

Поступила в редакцию 26 октября 2017 г.

INFORMATION IN ENGLISH

MAKING USE OF ALGORITHM OF BITMAP INDEXES TO INCREASE EFFICIENCY OF THE SEARCH QUERIES PROCESSING LOW SELECTIVITY DATA

Tat'yana N. Nosova

Assistant Professor, the Institute of Power engineering and automated systems, the Department of Informatics and information security, Nosov Magnitogorsk State Technical University, Magnitogorsk, Russia.

Ol'ga B. Kalugina

Ph.D. in Engineering, Assistant Professor, the Institute of Power engineering and automated systems, the Department of Informatics and information security, Nosov Magnitogorsk State Technical University, Magnitogorsk, Russia.

Query optimization is an important part of operation of any application interacting with databases. The most effective method to accelerate the search queries is to use index structures of different kinds. It is known that queries, which use the indexes (clustered and not clustered), are effective, in this case, when columns contain low percentage of repeating values. If indexable data are not selective, use of the majority of types of indexes is not effective. The main objective of the performed work is to extend the opportunities of technologies in the environment of SQL Server MS for creation of indexes and to increase the productivity of search queries. To achieve this purpose, the research group developed a Net-application in a programming language of the high level C# using an algorithm of formation of bit scales for processing of columns of relational tables with a large count of the duplicated values. The article offers a review of the main existing methods increasing the efficiency of execution of requests and also the types of the index structures used in different database management systems. The article contains some examples of application actions in selection of values with bit indexes. Testing of the created software product on tables with different cardinality makes it possible to draw a conclusion about the considerable saving of time of data handling in case of using bit indexes in comparison with other search algorithms.

Keywords: SQL query optimization, index of data, binary index, dictionary of data.

REFERENCES

1. Stain D.A., Hasovskih V.P. Methods of optimization and increase in access efficiency to data in management information systems of organization. *Fundamentalnyie issledovaniya* [Fundamental research], 2014, no. 12-10, pp. 2114-2119. (In Russian)
2. Optimization of productivity of execution of queries (SQL Server Compact). Microsoft. Developer Network [Digital resource]. Access mode: https://msdn.microsoft.com/ru-ru/library/ms172984.aspx. The title from the screen. (Date of the address: 10.10.2017).
3. Phrolov K.M., Knjazev V.N. Query tuning to databases on the basis of the index mechanism. *Sbornik nauchnykh statey XII mezhdunarodnoy nauchno-tekhnicheskoy konferentsii «Novye informatsionnye tekhnologii i sistemy»* [Collection of scientific articles of the XII international scientific and technical conference "New Information Technologies and Systems"], 2015, pp. 201-204. (In Russian)
4. Typical reasons of no optimal operation of queries and methods of optimization // Information and technological attending of users 1C: Enterprise [Digital resource]. Access mode: https://its.1c.ru/db/metod8dev#content:5842:hdoc. The title from the screen. (Date of the address: 10.10.2017).
5. Optimizing Database Structure [Digital resource]. Access mode: https://dev.mysql.com/doc/refman/5.7/en/optimizing-database-structure.html. The title from the screen. (Date of the address: 10.09.2017).
6. Petuhov I.S. Algorithm of determination of necessary indexes for query tuning with connection of two tables in MYSQL (INNODB) DBMS. *Nauchnyiy vestnik GosNII GA*. [Scientific bulletin of the state scientific research institute of civil aviation (GosNII GA)], 2017, no. 16, pp. 98-107. (In Russian)
7. Riggs S., Krossing X. PostgreSQL 9: Administration Cookbook. BIRMINGHAM – MUMBAI, Packt Publ., 2013, 368 p.
8. Tarasov C.V. *SUBD dlya programmista. Bazy dannykh iznutri* [The DBMS for the programmer. Databases from within.] Moscow, Solomon-Press Publ., 2015, 320 p. (In Russian)
9. Tumanov V.E. *Proektirovanie khranilisch dannykh dlya prilosheniy system delovoy osvedomlennosty* [Design of data stores for applications of systems of business awareness] (Business Intelligence Systems). Moscow, Intuit Publ., 2016, 958 p. (In Russian)
10. SQL Server 2000[Digital resource]. Access mode: http://www.intuit.ru/studies/courses/68/68/lecture/2016?page=3. The title from the screen. (Date of the address: 10.10.2017).
11. Microsoft. Developer NetWork [Digital resource]. Access mode: https://msdn.microsoft.com/ru-ru/library/ms175049(v=sql.120).aspx. The title from the screen. (Date of the address: 10.09.2017).
12. Barrie X. The Firebird Book: A Reference for Database Developers. Apress Publ., 2004, 1128 p.
13. Henderson K. The Guru's Guide to SQL Server Architecture and Internals. Addison – Wesley Publ., 2003, 685 p.
14. Microsoft academy: The distributed bases and data stores [Digital resource]. Access mode: http://www.intuit.ru/studies/courses/1145/214/lecture/5523?page=2 intuit. The title from the screen. (Date of the address: 10.10.2017).
15. Skit J. C# for professionals: programming subtleties. Moscow, Williams Publ., 2014, 610 p.

Носова Т.Н., Калугина О.Б. Использование алгоритма битовых шкал для увеличения эффективности поисковых запросов, обрабатывающих данные с низкой избирательностью // Электротехнические системы и комплексы. 2018. № 1(38). С. 63-67. https://doi.org/10.18503/2311-8318-2018-1(38)-63-67

Nosova T.N., Kalugina O.B. Making Use of Algorithm of Bitmap Indexes to Increase Efficiency of the Search Queries Processing Low Selectivity Data. *Elektrotekhnicheskie sistemy i kompleksy* [Electrotechnical Systems and Complexes], 2018, no. 1(38), pp. 63-67. (In Russian). https://doi.org/10.18503/2311-8318-2018-1(38)-63-67