**Ландовский Владимир Владимирович,**
кандидат технических наук,
ФГБОУ ВО Новосибирский государственный
технический университет, г. Новосибирск

# АЛГОРИТМ ПОИСКА ПО НАБОРУ СЛОВ

**Аннотация**: в статье рассматривается решение задачи полнотекстового поиска элементов справочника по заданному набору слов. Предложены структуры данных и алгоритм, позволяющий найти заданное число наиболее подходящих элементов. Приведены результаты тестирования работы алгоритма.

**Ключевые слова**: полнотекстовый поиск, поиск по словам, поиск с индексацией, алгоритм поиска.

Пусть имеется справочник, каждый элемент которого описывается большим количеством свойств, а каждое свойство принимает широкое множество значений. Под справочником понимается множество экземпляров некоторой сущности, работа с которым осуществляется в рамках реляционной модели [1]. Оператор вводит произвольную текстовую строку и в режиме реального времени (после ввода каждого очередного слова или даже символа) видит подборку элементов справочника наиболее похожих на введенный текст. Очевидно, что время поиска должно быть минимальным, поиск должен уложиться в доли секунды. Опечатки при написании слов выходят за рамки данной задачи. В рамках рассмотренного алгоритма считается, что слова написаны без ошибок.

Обеспечить приемлемую производительность поиска без предварительного формирования текстового описания и разбора его на слова невозможно. На рис. 1 представлен фрагмент базы данных. Основная таблица справочника – *catalog*, таблицы, соответствующие свойствам элементов

справочника, на рисунке отсутствуют, так как точное представление о структуре элемента не имеет значения для решаемой задачи. Предполагается, что поле *text* содержит полное текстовое описание элемента, это поле введено для удобства изложения. На практике полное описание вполне допустимо получать запросом.

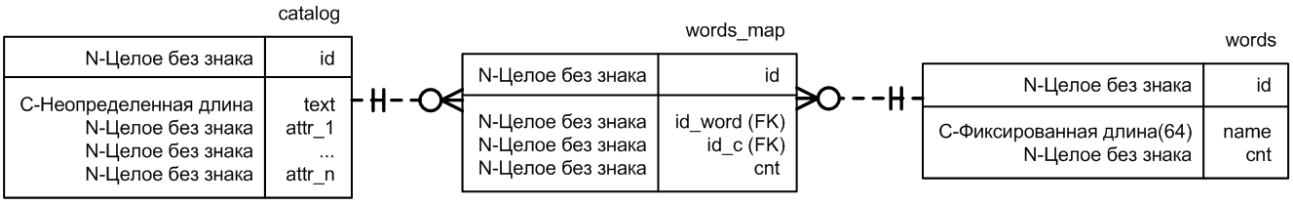


Рисунок 1

Таблица *words* содержит слова, встретившиеся в описании элементов справочника, поле *name* хранит само слово, а *cnt* – общее количество появлений данного слова в справочнике. Таблица *words_map* служит для реализации связи многие ко многим между элементами справочника и словами. Поле *cnt* в этой таблице определяет вес связи, он может определяться как количество вхождений слова в описание конкретного элемента. Другой, более гибкий, подход к определению веса связи – учитывать свойство, в котором встретилось слово, например, слово в наименовании будет иметь больший вес в сравнении с другими свойствами.

Формирование таблиц *words* и *words_map* происходит при изменении таблицы *catalog*. Если предусмотрена процедура загрузки справочника, то эти таблицы будут созданы в процессе загрузки. Операции добавления, удаления и редактирования данных в *catalog* должны корректировать данные в *words* и *words_map*. Если считать, что в нормальном рабочем режиме изменения в *catalog* вносятся редко, то время создания и изменения вспомогательных таблиц не представляет интереса.

Пусть множество слов поисковой строки обозначено *P*, а их количество – *np*. Для хранения количества совпавших слов в оперативной памяти формируется массив целых чисел *Ms*. Размер этого массива *ns* равен количеству элементов в справочнике. Номер элемента в массиве соответствует номеру строки в таблице *catalog*, значение элемента массива – количество совпавших слов. Такой способ хранения данных позволит за O(1) обращаться к количеству совпавших слов для *i*-го элемента. Заполнение *Ms* осуществляется следующим образом.

```
Для i от 0 до ns
    Ms[i]←0
MaxCnt←0
Для i от 0 до np
    R←GetWcnt(P[i])
    Для каждого r из R
        Ms[r.id]←Ms[r.id] + r.cnt
        Если MaxCnt < Ms[r.id] Тогда
            MaxCnt←Ms[r.id]
GetWcnt(P[i]):
SELECT id_c AS id, cnt FROM words_map
WHERE id_word = (SELECT id FROM words WHERE name = P[i])
```

Время подготовительной работы по формированию *Ms* состоит из двух частей. Первая – это время присвоения нулевых начальных значений, она линейна относительно *ns*. Вторая – поиск и суммирование совпадений, она линейна относительно общего количества вхождений всех слов из *P* во все элементы *catalog*. Кроме того она зависит от скорости поиска слов в таблице *words* на уровне СУБД.

Теперь необходимо запомнить ненулевые элементы *Ms* и их позиции, чтобы не потерять их в процессе сортировки. Переменная *MaxCnt* хранит значение максимального количества вхождений (максимального суммарного веса) слов. С её помощью, принимая во внимание, что это значение невелико, можно организовать сортировку за линейное по отношению к количеству сортируемых объектов время.

Пусть *MLcnt* – массив связных списков чисел, размерность массива: *MaxCnt, Ret* – список чисел для формирования результата. Тогда сортировку, ограниченную возвращением *k* идентификаторов строк с наибольшим совпадением слов, можно выполнить в соответствии со следующим алгоритмом, базирующемся на карманной сортировке [2].

```
Для i от 0 до ns
        Если Ms[i] > 0
                MLcnt[Ms[i]-1].Добавить(i)
nk←0
Для i от MaxCnt до 1
        Для каждого id из MLcnt[i-1]
                Ret.Добавить(id)
                nk←nk+1
                Если nk= k
                        Вернуть Ret
```

Временная сложность сортировки O(*ns*+*k*)

Оптимизация: кэширование популярных слов

Наибольшее замедление первой фазы алгоритма связано со словами, встречающимися в большом количестве строк, эти слова можно условно назвать популярными. Объясняется это в частности необходимостью передачи больших объемов данных между приложением и СУБД при выполнении запроса в функции GetWcnt(P[i]).

Ускорить обработку популярных слов можно следующим образом. Пусть в массиве *PopWords* размера *nw* хранятся целые числа. Если *i* соответствует номеру слова в таблице *words*, то *PopWords*[*i*] содержит неотрицательное число, которое является первым индексом в двумерных массивах *PopWordsMap* и *PopWordsMapCnt*. Размерность этих массивов равна количеству популярных слов на максимально возможное количество строк, соответствующих слову. Пусть количество популярных слов обозначается *npopw*, выбирается оно максимальным исходя из объемов доступной оперативной памяти. Максимально возможное количество соответствий определяется конкретными данными, может быть заранее подсчитано, однако худшая оценка это *ns*. В реальности она достижима, если все элементы справочника характеризуются некоторым свойством, одно из значений которого заметно доминирует над остальными. Массив *PopWordsMap*[*PopWords*[*i*]], где *i* идентификатор некоторого слова в базе данных, содержит идентификаторы элементов справочника соответствующих слову. Идентификаторы размещаются в массиве с нулевого индекса, если некоторый *PopWordsMap*[*PopWords*[*i*]][*j*] меньше нуля, то элемент в позиции (*j*–1) следует считать последним заполненным, а следующие позиции не рассматривать. Во втором массиве хранятся соответствующие количества вхождений (или веса), то есть *PopWordsMapCnt*[*PopWords*[*i*]][*j*] интерпретируется как количество вхождений слова с идентификатором *i* в строку с идентификатором *PopWordsMap*[*PopWords*[*i*]][*j*]. Эти данные можно было бы хранить в одном массиве – последовательно парами: первое число идентификатор, следующее – количество. Разделение оправдано, с одной стороны, тем, что массив – это последовательный единый фрагмент памяти, разделение облегчит размещение данных. С другой стороны, количество байт для хранения количества вхождений (тип данных массива *PopWordsMapCnt*) можно сократить до одного, что существенно уменьшит общие требования к памяти.

Построение такой вспомогательной структуры данных осуществляется единожды при запуске поисковой программы. Алгоритм первой фазы поиска с учётом оптимизации приведен ниже.

```
Для i от 0 до ns
    Ms[i]←0
MaxCnt←0
Для i от 0 до np
    wid←GetWid(P[i])
    widx←PopWords[wid]
    Если widx >0 Тогда
        j←0
        sid←PopWordsMap[widx][j]
        Пока sid > 0
            Ms[sid]←Ms[sid] + PopWordsMapCnt[widx][j]
            Если MaxCnt < Ms[sid] Тогда
                MaxCnt←Ms[sid]
            j←j+1
            sid←PopWordsMap[widx][j]
    Иначе
        R←GetWcntById(wid)
        Для каждого r из R
            Ms[r.id] ←Ms[r.id] + r.cnt
            Если MaxCnt<Ms[r.id] Тогда
                MaxCnt←Ms[r.id]
```

GetWid(P[i]): SELECT id FROM words WHERE name = P[i]

GetWcntById(i): SELECT id_c AS id, cnt FROM words_map WHERE id_word = i

Реализация алгоритма выполнена в виде оконного приложения для компьютеров под управлением операционной системы семейства Windows. Среда разработки – Microsoft Visual Studio, язык C#, платформа .Net 4.5 [3]. Окно программы показано на рис. 2, в верхней части окна расположено поле ввода поисковой строки и кнопка поиска, ниже расположен список результатов.

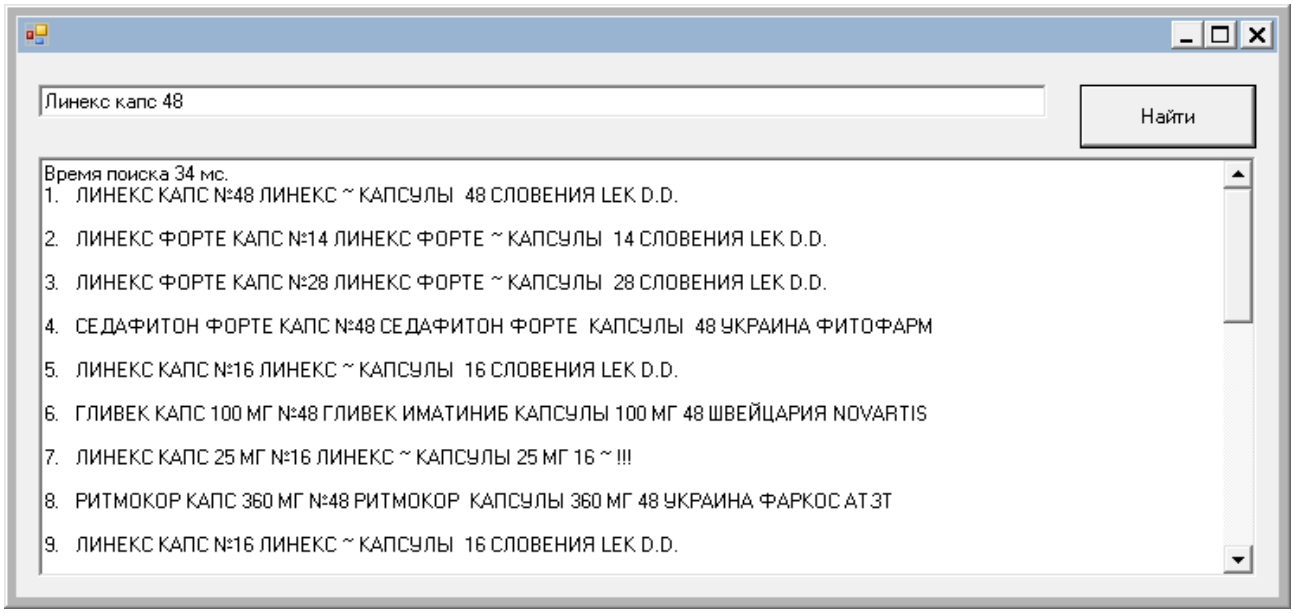


Рисунок 2

Для хранения данных использована СУБД PostgreSQL версии 9.5 [4]. Работа с базой данных осуществляется с помощью библиотеки npgsql версии 3.1.5.

Тестирование алгоритма проведено на примере справочника лекарственных препаратов. Элемент справочника описывается следующим набором свойств: наименование (название препарата), торговое наименование, международное непатентованное наименование, страна производитель, фирма производитель, лекарственная форма, дозировка, номер упаковки, единица измерения.

Справочник содержит сведения о 113 тысячах препаратов. В качестве слова рассматривается любая последовательность букв и цифр отделенная прочими символами. После разбора на слова в таблице *words* оказалось 36833

записей, в таблице *words_map* – 1598783 записей. Наиболее популярное слово имеет порядка 50 тысяч связей, т. е. встречается в описании 50000 элементов справочника. Пятидесятое в порядке убывания популярности имеет уже 5 тысяч связей, двухсотое менее одной тысячи. Время старта программы при кэшировании в памяти двухсот популярных слов порядка двух секунд.

Чтобы продемонстрировать преимущества от кэширования популярных слов дополнительно осуществлялся поиск без кэширования – для всех слов осуществлялись запросы к таблицам *words* и *words_map*.

Количество выводимых результатов в том и другом случае равно двадцати – выводится двадцать элементов справочника, в порядке убывания количества совпавших слов.

Для сравнения приводится время поиска выполненного напрямую средствами СУБД. Пример запроса при поиске двух слов, условно обозначенных слово1 и слово2, приведен ниже.

```
SELECT id, text, SUM(numb)
FROM(
SELECT id, text, 1 as numb
  FROM catalog
 WHERE text like '%слово1%'
 UNION ALL
SELECT id, text, 1 as numb
  FROM catalog
 WHERE text like '%слово2%'
) as un
GROUP BY id, text
ORDER by SUM DESC
limit 20
```

Результаты приведены в таблице.

| Поисковая строка | Время алгоритма, мс | Время алгоритма без популярных слов, мс | Время SQL запроса, мс |
|---|---|---|---|
| ЛИНЕКС КАПСУЛЫ | 7 | 40 | 380 |
| ЛИНЕКС КАПСУЛЫ 1 МГ | 15 | 200 | 760 |
| ЦВЕТКИ БОЯРЫШНИКА | 5 | 12 | 350 |

*Список литературы:*

1. Карпова Т.С. Базы данных: модели, разработка, реализация – СПб. : Питер, 2001. – 304 с.

2. Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д. Структуры данных и алгоритмы. Пер. с англ.: Уч. пос. – М.: Издательский дом "Вильямс", 2000. – 384 с.

3. Петцольд Ч. Программирование для Microsoft Windows на C# В 2-х томах: Пер. с англ. – М.: Издательско-торговый дом "Русская Редакция", 2002.

4. Стоунз Р., Мэтью Н. PostgreSQL. Основы. – Пер. с англ. – СПб: Символ Плюс, 2002. – 640 с.