УДК 519.86
ББК 22.18

ОПТИМИЗАЦИЯ СТРУКТУРЫ БАЗЫ ДАННЫХ РЕАЛЬНОГО ВРЕМЕНИ

Мирошник С. Н.[1, a], Гончар Д. Р.[2, a, b]
Фуругян М. Г.[3, a, c]
([a]*Вычислительный центр им. А.А. Дородницына ФИЦ «Информатика и управление» РАН,*
[b]*Московский физико-технический институт (государственный университет),*
[c]*Московский государственный университет им. М.В. Ломоносова, Москва*)

Исследуется задача оптимизация структуры базы данных реального времени. Разработаны эвристические алгоритмы решения данной задачи и алгоритм, основанный на сведении ее к задаче булевого программирования.

Ключевые слова: системы управления базами данных, системы реального времени, эвристические алгоритмы, оптимизация.

1. Введение

Одной из основных проблем проектирования баз данных (БД) реального времени является проблема минимизации избыточности информации. Теоретические основы создания и оптимизации БД содержатся в [1, 6]. Настоящая работа является

[1] *Сергей Николаевич Мирошник, научный сотрудник, кандидат физико-математических наук* (*rtsccas@ya.ru*).
[2] *Дмитрий Русланович Гончар, старший научный сотрудник, кандидат технических наук* (*trpl@ya.ru*).
[3] *Меран Габибуллаевич Фуругян, заведующий сектором, кандидат физико-математических наук, доцент* (*rtsccas@ya.ru*).

продолжением работ [2–4, 5], посвященных методам создания соответствующей структуры БД, в которой избыточность информации сведена к минимуму. Этой проблеме посвящено большое количество публикаций, в которых основное внимание уделено технической части создания подобных БД, но не связанных с использованием информации в БД для решения задач в реальном времени. В данной работе основное внимание уделено математической стороне проблемы. В работах [2–4, 5] используется понятие избыточности информации трех типов, а именно межфайловая, внутрифайловая и внутримодульная избыточности. В работах [2, 4] подробно даны определения этих понятий, а также формулы, с помощью которых эти избыточности вычисляются. В зависимости от типов задач разработаны эвристические алгоритмы, минимизирующие разные типы избыточностей. Так, например, в работе [4] рассматриваются задачи, в которых неиспользуемая информация сравнима с используемой. Предлагается алгоритм создания БД, в которой минимизируется внутримодульная избыточность информации. В данной работе, так же как в работе [2], основное внимание уделено минимизации межфайловой и внутрифайловой избыточностей. Предлагается принципиально новый, более совершенный, алгоритм создания БД по сравнению с аналогичными алгоритмами, предложенными в [2]. Эти результаты используются в разрабатываемой авторами инструментальной системе автоматизации проектирования систем реального времени, в которой эта БД является одной из составляющих элементов. Данная САПР СРВ может быть использована при решении задач, возникающих при лётных экспериментах, экологическом или экономическом мониторинге и в ряде других областях.

2. Постановка задачи

Имеется n независимых друг от друга программных модулей $i = 1, ..., n$. Для их выполнения требуются данные, содержащиеся в файле Ф, который состоит из полей $Ф_1, Ф_2, \ldots, Ф_r$. При выполнении каждого модуля используется часть полей $Ф_j$ файла Ф. Для заданного числа K строятся файлы $F_1, F_2, \ldots, F_k$,

$k \le K$, каждый из которых содержит некоторые поля Φ_j файла Φ, причем для каждого модуля i некоторый файл $F_{j(i)}$ должен содержать все поля Φ_j, необходимые для его работы. Суммарная длина файлов F_i не должна превышать заданной величины W. Внутрифайловая избыточность входной информации определяется как величина $I_1 = \sum_{i=1}^{n}(|F_{j(i)}| - |\overline{F}_{j(i)}|)$, где $|F_{j(i)}|$ – длина файла $F_{j(i)}$, $|\overline{F}_{j(i)}|$ – суммарная длина полей файла $F_{j(i)}$, используемых работой i. Межфайловая избыточность базы данных определяется как величина $I_2 = \sum_{j=1}^{r}(K_j - 1)|\Phi_j|$, где K_j – число файлов F_i, в которых содержится поле Φ_j, $|\Phi_j|$ – длина поля Φ_j. Задача заключается в создании для заданных K и W такого набора файлов $F_1, F_2, \ldots, F_k$, $k \le K$, для которого

(1) $\sum_{j=1}^{k}|F_j| \le W$

и величина I_1 минимальна. Впервые авторами эта постановка была сформулирована в работе [2]. Если решений более одного, то из их числа выбирается такое, при котором величина I_2 минимальна. В качестве критерия оптимальности можно взять линейную комбинацию величин I_1 и I_2, например, $I_1 + I_2$.

Отметим, что решение задачи в разрабатываемой авторами инструментальной системе автоматизации проектирования систем реальном времени разделяется на два больших этапа.

A. Предварительный этап (не в реальном времени), на котором в том числе осуществляется формирование групп близких модулей.

B. После этого решается задача в реальном времени, используя выполненную на предварительном этапе оптимизацию расположения полей в файлах.

3. Сведение к задаче булевого программирования

Пусть v_{pj}, $p = 1, 2, \ldots, n$, $j = 1, 2, \ldots, r$, – массив, состоящий из нулей и единиц, причем $v_{pj} = 1$, если для выполнения модуля p требуется поле Φ_j, и $v_{pj} = 0$ в противном случае.

Введем переменные x_{ij}, $i = 1, 2, \ldots, k$; $j = 1, 2, \ldots, r$, и y_{ip}, $i = 1, 2, \ldots, k$; $p = 1, 2, \ldots, n$, принимающие значения 0 и 1, причем $x_{ij} = 1$, если файл F_i содержит поле Φ_j, и $x_{ij} = 0$ в противном случае; $y_{ip} = 1$, если для работы модуля p требуется файл F_i, и $y_{ip} = 0$, в противном случае.

Сформулированная задача может быть записана в виде следующей задачи булевого программирования [2]:

$$(2)\quad \min_{x_{ij}, y_{ip}} \sum_{p=1}^{n}\sum_{i=1}^{k}\left\{ y_{ip}\left[\sum_{j=1}^{r}(x_{ij} - v_{pj})\cdot\left|\Phi_j\right|\right]\right\};$$

$$(3)\quad \min_{x_{ij}} \sum_{j=1}^{r}\left[\left(\sum_{i=1}^{k} x_{ij}\cdot\left|\Phi_j\right|\right) - \left|\Phi_j\right|\right];$$

$$(4)\quad \sum_{i=1}^{k}\sum_{j=1}^{r} x_{ij}\cdot\left|\Phi_j\right| \le W;$$

$$(5)\quad \sum_{i=1}^{k} y_{ip} = 1, \quad p = 1, 2, \ldots, n;$$

$$(6)\quad \begin{array}{l} y_{ip}(x_{ij} - v_{pj}) \ge 0; \quad i = 1, 2, \ldots, k; \\ p = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, r. \end{array}$$

Здесь x_{ij}, y_{ip} принимают значения 0 и 1. Число переменных в данной задаче равно $k(r + m)$, а число ограничений – $(nkr + 1)$.

Условие (2) соответствует минимизации величины i_1, условие (3) – минимизации величины i_2, условие (4) – условию (1), условие (5) приписывает каждому модулю $p \in n$ некоторый файл f_i. Благодаря условию (6) у каждого модуля $p \in n$ соответствующий ему файл f_i будет содержать все необходимые поля Φ_j. Сначала должна быть решена задача минимизации (2) при ограничениях (4)–(6). Затем на полученном множестве решений (если решение не однозначно) решается задача минимизации (3).

4. Эвристический алгоритм создания структуры БД

Будем предполагать, что все поля пронумерованы натуральным рядом чисел 1, …, r. Запись модуля M_i есть набор полей $\{\phi\}_i$, идущих подряд из набора ϕ_1, …, ϕ_r. Таким образом, в записи модуля M_i есть номер первого поля a_i и номер последнего поля b_i. Длина записи модуля M_i есть $l_i = b_i - a_i + 1$. Записи разных модулей могут использовать одинаковые поля. Природа полей и их длины не рассматриваются. В данной работе в записи модулей входят только используемые поля, неиспользуемые поля не рассматриваются, так как они не влияют на работу алгоритма. Основная идея минимизации избыточной информации состоит в том, чтобы определенным образом объединить модули в различные группы, в дальнейшем оформляемых в виде файлов со своими атрибутами с последующим вычислением I_1 и I_2 (см. [3]). Это объединение осуществляется так, чтобы каждый модуль попал только в один файл. Длина L записи файла F определяется количеством различных полей всех модулей, образующих этот файл. В записи файла F есть первое поле A со своим номером и последнее B, $L = B - A + 1$. Теперь каждый модуль M для своей работы обращается ко всем полям записи своего файла F. В частности, различные модули могут использовать одинаковые поля файла. Вследствие разности длины l модуля M и длины L файла F образуются неиспользуемые этими модулями поля, $L \geq 1$. Эти неиспользуемые поля всех модулей файла F образуют внутрифайловую избыточность $I_1(F)$ файла F. Эта избыточная информация вычисляется по формуле

$$(7) \quad I_1(F) = L \cdot t - \sum_{j=1}^{t} l^j ,$$

где t – число модулей в файле F.

Естественно, полная внутрифайловая избыточность БД определяется по формуле

$$(8) \quad I_1 = \sum_{i=1}^{k} (L_i \cdot t_i - \sum_{j=1}^{t} l_i^j)$$

для всех файлов F_i, …, F_k.

Далее, среди полей записей файлов F_i есть повторяющиеся поля. Эти поля образуют межфайловую избыточность I_2 и вычисляются по формуле

$$(9) \quad I_2 = \sum_{i=1}^{k} L_i - r.$$

4.1. ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

Предложенные в [3] алгоритмы так же, как и в данной работе, основаны на различных способах объединения модулей в файлы. Главным в этом объединении есть определение понятия близости двух модулей. Для того чтобы сравнить алгоритмы из [3] и из настоящей работы, приведено описание краткой схемы алгоритма из [3].

Алгоритм из [3] имеет существенные недостатки. Модули объединяются в файлы на основе формальных признаков близости модулей, не учитывающих образующиеся внутрифайловые и межфайловые избыточности информации.

Схема алгоритма состоит в следующем. Строится распределение P повторяемости полей файла Ф. Вводится понятие опорного модуля, содержащее поле максимальной повторяемости ϕ^* и максимальной длиной l^*. Близкий к опорному модуль M длиной l должен содержать поле ϕ^* и $l \le l^*$. Перебором находим все модули, удовлетворяющие этим двум условиям. Найденные таким образом модули исключаются из $M_1, \ldots, M_n$ вместе со всеми значениями используемых ими полей. После этого строится новое распределение P повторяемости полей. Если найденное ранее поле ϕ^* в новом распределении P не изменилось, увеличиваем длину l^* на величину Δ и ищем новые модули длиной $l \le l^* + \Delta$ и содержащие поле ϕ^*. Увеличение длины l^* продолжается до тех пор, пока в очередном распределении полей P не изменится поле ϕ^*. Найденные таким образом модули образуют файл F. Процедура повторяется, пока все модули не будут распределены по файлам $F_1, \ldots, F_k$. Только после этого вычисляются I_1 и I_2 по формулам (7)–(9).

В данной работе предлагается принципиально другой алгоритм объединения модулей $M_1, \ldots, M_n$ в файлы $F_1, \ldots, F_k$. Этот

алгоритм использует другое понятие близости пары модулей M_i и M_j, а также близость модуля M к набору модулей $\{M\}$.

Введем понятие близости набора модулей $\{M\}_s$ и модуля M_{s+1}, где s – число модулей в наборе. Будем считать, что $\{M\}_s$ и M_{s+1} являются близкими, т.е. M_{s+1} может быть включен в набор модулей $\{M\}_s$, если будет выполнено определенное условие. Чтобы определить это условие, необходимо вычислить два вида избыточности информации.

1. Набору $\{M\}_s$ соответствует избыточность информации I_1^s. Включение M_{s+1} в $\{M\}_s$ изменяет внутрифайловую избыточность информации на величину $\Delta I_1^{s+1} = I_1^{s+1} - I_1^s$, где I_1^{s+1} соответствует избыточности информации набора модулей $\{M\}_{s+1}$.

2. Для модуля M_{s+1} и набора модулей $\{M\}_s$ вычислим количество совпадающих полей. Для этого воспользуемся модифицированной формулой (3). Обозначим I_2^{s+1} – количество совпадающих полей пары M_{s+1} и $\{M\}_s$.

Замечание. Формула (3) используется для вычисления избыточной информации набора файлов, а именно вычисления количества совпадающих полей разных файлов. В данном случае эта формула вычисляет значение I_2^{s+1} для определения близости пары M_{s+1} и $\{M\}_s$.

Определение. Модуль M_{s+1} и набор модулей $\{M\}_s$ являются близкими и M_{s+1} может быть включен в набор модулей $\{M\}_s$, если $\Delta I_1^{s+1} \le I_2^{s+1}$.

4.2. АЛГОРИТМ СОЗДАНИЯ СТРУКТУРЫ БД

Алгоритм использует определение близости модулей. Пусть приведенным ниже алгоритмом построен набор близких модулей $\{M\}_s$, s – число модулей. Рассмотрим модуль M_{s+1}, l_{s+1} – его длина. Относительно этого модуля требуется определить: $M_{s+1} \subset \{M\}_s$ или $M_{s+1} \not\subset \{M\}_s$.

Внутрифайловая избыточная информация I_1^s в наборе модулей $\{M\}_s$ вычисляется с помощью формулы (7): $I_1^s = L_s \cdot s - \sum_{i=1}^{s} l_i$, где L_s длина набора модулей $\{M\}_s$.

1. Вычислим ΔI_1^{s+1}. Для набора модулей $\{M\}_{s+1}$ избыточная информация есть

$$I_1^{s+1} = L_{s+1} \cdot (s+1) - \sum_{i=1}^{s+1} l_i ,$$

где L_{s+1} длина набора модулей $\{M\}_{s+1}$. Дополнительная избыточная информация ΔI_1^{s+1} есть $\Delta I_1^{s+1} = I_1^{s+1} - I_1^s$, или

$$\Delta I_1^{s+1} = L_{s+1} \cdot (s+1) - (\sum_{i=1}^{s} l_i + l_{s+1}) - L_s \cdot s + \sum_{i=1}^{s} l_i .$$

Окончательно

(10) $\Delta I_{s+1} = (L_{s+1} - L_s) \cdot s + L_{s+1} - l_{s+1}$.

Здесь $L_s = b_s^* - a_s^* + 1$, $l_{s+1} = b_{s+1} - a_{s+1} + 1$, $L_{s+1} = b_{s+1}^* - a_{s+1}^* + 1$, $b_{s+1}^* = \max(b_s^*, b_{s+1})$, $a_{s+1}^* = \min(a_s^*, a_{s+1})$, $b_s^* = \max\{b\}_s$, $a_s^* = \min\{a\}_s$.

2. Определим I_2^{s+1}. Для пары M_{s+1} и $\{M\}_s$ вычислим по формуле (9) избыточную информацию I_2^{s+1}. Получаем

(11) $I_2^{s+1} = L_s + l_{s+1} - (b_{s+1}^* - a_{s+1}^*)$.

Теперь согласно определению условие $\Delta I_1^{s+1} \le I_2^{s+1}$ о вхождении $M_{s+1} \subset \{M\}_s$ записывается так:

(12) $(L_{s+1} - L_s) \cdot (s+1) \le 2 \cdot l_{s+1} - (b_{s+1}^* - a_{s+1}^*)$.

Если условие (12) не выполняется, то M_{s+1} вместе с модулями, не вошедшими в набор $\{M\}_s$, могут образовывать другие группы похожих модулей.

4.3. ВЫЧИСЛИТЕЛЬНАЯ ЧАСТЬ АЛГОРИТМА

Для поиска близких модулей с целью формирования группы модулей $\{M\}$ требуется многократный перебор всех модулей. Существует одна особенность при реализации перебора. На некотором шаге перебора может оказаться, что длина L группы

$\{M\}$ увеличилась. Отсюда следует, что ранее отвергнутые модули могут оказаться близкими к $\{M\}$. Поэтому для поиска всех модулей, близких к набору $\{M\}$, требуется многократный перебор модулей. Естественно, из перебора исключаются те модули, которые уже попали в группу $\{M\}$.

Поиск первой группы близких модулей реализуется с помощью нескольких вычислительных шагов.

1. Путем перебора находим модуль M, для которого $l = \max(l_1, \ldots, l_n)$.

2. Перебором ищем модуль, близкий к $\{M\}$. Заметим, что при этом переборе каждый следующий рассматриваемый модуль должен быть близок к уже найденной ранее группе модулей.

3. Если после первого перебора модулей увеличилась длина L группы модулей $\{M\}$, то перебор модулей для поиска близких к набору $\{M\}$ повторяется.

4. Перебор модулей для поиска близких к набору $\{M\}$ заканчивается, если после очередного перебора модулей длина L построенной группы $\{M\}$ не изменилась.

5. Построенная группа модулей оформляется как файл F с соответствующими характеристиками: длиной L, списком модулей и их количеством, внутрифайловой избыточностью $I_1(F)$.

6. Модули файла F исключаются из набора модулей $M_1, M_2, \ldots, M_n$ и не используется повторно для построения последующих файлов.

7. Начиная с шага 1 строятся все файлы $F_1, F_2, \ldots, F_k$.

Алгоритм заканчивает работу, когда все модули будут распределены по файлам.

Замечания

1. В результате объединения модулей в файлы могут остаться модули, не вошедшие ни в один файл. Эти «одиночные» модули образуют отдельные файлы. В частности, «одиночным» модулем может оказаться модуль, использующий только одно поле. Этот модуль также может быть оформлен как отдельный файл.

2. Следует отметить возможную неоднозначность выбора первого модуля, а также неоднозначность выбора последующих близких модулей. Отсюда может быть построено множество

различных вариантов файлов и соответствующих им БД. В данной работе предлагается построить только один вариант набора файлов.

Вычисление внутрифайловой избыточности осуществляется по формуле $I_1 = \sum_{i=1}^{k} I_1(F_i)$.

Для найденного набора файлов $\{F\}_k$ определяется внутрифайловая избыточность I_1 и межфайловая I_2. Число $I = I_1 + I_2$ является показателем качества построенной БД. Вообще, учитывая замечание 2, из множества чисел I для каждого варианта БД можно выбрать наилучшую БД.

Вычислительная сложность алгоритма 2 есть $O(n^3r)$, вычислительная сложность алгоритма 1 есть $O(n^2r^3)$.

Для конкретной задачи значение для I_1 и I_2 можно сравнить с их наибольшими значениями. Пусть все модули объединены в один файл. Тогда $I_2 = 0$, а I_1 принимает максимальное значение. Если каждый модуль образует отдельный файл, то $I_1 = 0$ и I_2 принимает максимальное значение.

4.3. ВЫЧИСЛИТЕЛЬНАЯ ЧАСТЬ АЛГОРИТМА

На приводимом ниже примере показывается работа описанных в этом разделе алгоритмов.

Пример задан в виде таблицы.

Таблица. Пример работы описанных в разделе алгоритмов

№ модуля	№ поля								
	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1		
2	1	1	1						
3			1	1	1				
4			1	1	1	1	1	1	
5					1	1	1	1	
6		1	1	1					
7						1	1	1	1

В этом примере число модулей $n = 7$, число полей $r = 9$. Модули не содержат неиспользуемые поля, так как они не влияют на работу алгоритма и, если нужно, их можно учесть на последней стадии работы алгоритма.

Алгоритм 1.

1.1. Из распределения P повторяемости полей получаем $ф^* = 3$. Опорный модуль M_1. F_1 : 1, 2, 3, 6, $L_1 = 7$, $I_1(F_1) = 12$.

Исключим модули 1, 2, 3, 6.

1.2. Из нового распределения P получаем $ф^* = 5$. Опорный модуль M_4. F_2 : 4, 5, $L_2 = 6$, $I_1(F_2) = 2$.

1.3. F_3 : 7, $L_3 = 4$, $I_1(F_3) = 0$.

Получаем $I_1 = 14$, $I_2 = 8$, $I = 22$.

Алгоритм 2.

2.1. F_1 : 1, 4, $L_1 = 8$, $I_1(F_1) = 3$.

2.2. F_2 : 5, 7, $L_2 = 5$, $I_1(F_2) = 2$.

2.3. F_3 : 2, 6, $L_3 = 4$, $I_1(F_3) = 2$.

2.4. F_4 : 3, $L_4 = 3$, $I_1(F_4) = 0$.

Получаем

$I_1 = 7$, $I_2 = 11$, $I = 18$.

Значение I для второго алгоритма меньше первого, и потому БД, построенная вторым алгоритмом, лучше. Для сравнения: $\max I_1 = 33$, $\max I_2 = 21$.

Литература

1. ДЕЙТ К. ДЖ. *Введение в системы баз данных*. – 8-е изд. – М.: Вильямс, 2006.
2. МИРОШНИК С.Н., ФУРУГЯН М.Г. *Оптимизация структуры базы данных реального времени* // Некоторые алгоритмы планирования вычислений в многопроцессорных системах. – М.: ВЦ РАН, 2012. – С. 24–37.
3. МИРОШНИК С.Н. *Алгоритм оптимизации базы данных реального времени для двух файлов* // Некоторые алгоритмы планирования вычислений и оптимизации баз данных в многопроцессорных системах. – М.: ВЦ РАН, 2013. – С. 41–48.
4. МИРОШНИК С.Н. *Алгоритм оптимизации базы данных реального времени для неиспользуемых полей модулей* // Некоторые алгоритмы планирования вычислений и методы многокритериальной оптимизации для многопроцессорных систем. – М.: ВЦ РАН, 2014. – С. 32–40.
5. МИРОШНИК С.Н. *Алгоритм оптимизации структуры базы данных реального времени с минимальной избыточностью информации* // Некоторые алгоритмы составления расписаний в многопроцессорных системах. – М.: ВЦ РАН, 2015. – С. 25–34.
6. ULMAN L. *PHP6 and MYSQL for dynamic web sites*. – Peachpit Press, 2007.

REAL-TIME DATABASE STRUCTURE OPTIMIZATION

Sergey Miroshnik, Dorodnicyn Computing Centre of FRC «Informatics and Control» of RAS, Moscow, Cand. Sc. (rtsccas@ya.ru).
Dmitry Gonchar, Dorodnicyn Computing Centre of FRC «Informatics and Control» of RAS, Moscow, Moscow Institute of Physics and Technology (State University), Moscow, Cand. Sc. (trpl@ya.ru).
Meran Furugyan, Dorodnicyn Computing Centre of FRC «Informatics and Control» of RAS, Moscow, Lomonosov Moscow State University, Moscow, Cand. Sc. (rtsccas@ya.ru).

Abstract: *The problem of real-time database structure optimization is considered. The objective is to minimize the information redundancy with respect to real-time usage. There is a set of active processes, each process uses several data fields from a given database. The fields should be divided into a set of files such that each process need not use more than one file for normal operation. Different measures are proposed for intra-file, inter-file and inter-process redundancies. We focus on the problem of inter- and intra-file redundancy minimization which can be reduced to a Boolean programming problem. We propose two algorithms for the database structure optimization problem. The first algorithm is based on the Boolean programming reduction and the second one is heuristics with a polynomial computational complexity. An illustrative example is provided. The algorithms are included into a CAD for real-time systems which can be used for flight experiments, ecological monitoring and other fields.*

Keywords: database management system, real-time systems, heuristic algorithms, optimization.

Статья представлена к публикации
членом редакционной коллегии А.А. Лазаревым.

Поступила в редакцию 28.01.2016.
Опубликована 31.03.2017.