Методы организации иерархий в реляционных базах данных

1. Метод реляционного моделирования иерархии списком смежности 1

2. Метод материализацией пути 5

3. Метод вложенных множеств 10

4. Метод вложенных интервалов 21

5. Метод объединяющий материализацию пути и список смежности 36

6. Метод транзитивного замыкания 40

БИБЛИОГРАФИЧЕСКИЙ СПИСОК 42

1. Метод реляционного моделирования иерархии списком смежности

Список смежности является реляционным отображением представления деревьев в традиционном программировании. Основой данного метода является хранение прямых связей между *родителем* и его *сыновьями*. Каждый внутренний элемент древовидной иерархии обладает связью с элементом, который является его предшественником и находится на один уровень выше. Используя данное свойство иерархии можно организовать реляционную зависимость на уровне одной таблицы. В кортеж вводится поле, которое определяет предшественника данного элемента.

Реляционное отношение, описывающее дерево в рамках модели списка смежности.

Реляционное отношение, описывающее дерево в рамках модели вложенных множеств, строится по следующим принципам:

– Каждый элемент иерархии кодируется уникальным идентификатором для связи с другими реляционными отношениями. Данный идентификатор занимает поле id.

– Каждый элемент иерархии идентификатор своего родителя. Для этого используется: parent.

Таблица

Реляционное отношение для списка смежности

Поле	**Тип**	**Описание**
id	int4	Идентификатор элемента
parent	int4	Идентификатор предшествующего элемента

Операция добавления элемента в дерево.

Для добавления элемента в дерево, организованное как список смежности, необходимо знать идентификатор элемента, сыном которого будет новый элемент. Пусть *ID* есть идентификатор нового элемента, *PARENT_ID* есть идентификатор его родителя, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на вставку нового элемента будет выглядеть следующим образом:

insert into tree (id, parent) values (ID, PARENT_ID);

Операция удаления элемента из дерева.

Как было сказано ранее, при удалении элемента из дерева необходимо не только удалить элемент, но и перенастроить связи оставшихся элементов, для сохранения древовидной иерархии. Так, при организации дерева как списка смежности, необходимо проставить у сыновей удаляемого элемента нового родителя, а именно родителя удаляемого элемента. Пусть *ID* есть идентификатор удаляемого элемента, *PARENT_ID* есть идентификатор его родителя, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на удаление элемента будет выглядеть следующим образом:

update tree set parent=PARENT_ID where parent=ID;

delete from tree where id=ID;

Операция поиска родителей элемента.

Для нахождения родителей определенного элемента дерева при организации дерева как списка смежности необходимо выполнить ряд SQL-

запросов. С каждым SQL-запросом определяется родитель текущего элемента. Пусть *ID* есть идентификатор текущего элемента, *k* есть уровень текущего элемента, а *tree* является таблицей, в которой хранится дерево, тогда для поиска всех родителей элемента с идентификатором *ID* необходимо *k* раз рекурсивно вызвать следующий SQL-запрос:

select id, parent from tree where id = ID;

Для каждого последующего вызова данного запроса *ID* должно принимать значение *parent*, полученное в ходе выполнения предыдущего.

Операция поиска сыновей элемента.

Нахождение сыновей элемента дерева при его организации как списка смежности является достаточно простым и требует выполнения одного SQL-запроса. Пусть *ID* есть идентификатор элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск сыновей элемента будет выглядеть следующим образом:

select id from tree where parent=ID;

Операция поиска элементов поддерева.

Нахождение элементов поддерева с определенным элементом в качестве корня является трудоемкой операцией на дереве, организованном как список смежностей. Для решения задачи поиска элементов поддерева необходима рекурсия. На каждой итерации данной рекурсии мы вызываем SQL-запрос для получения списка сыновей текущего элемента, а затем вызываем этот же запрос для поиска сыновей по очереди для каждого элемента из данного списка. Начальным элементом для рекурсии является корень поддерева.

Операция перемещения поддерева.

Перемещение поддерева внутри иерархии в случае, когда дерево организовано как список смежностей, требует выполнения одного SQL-запроса. Пусть *ROOT_ID* идентификатор корня поддерева, *ID* идентификатор элемента, который в новом дереве будет являться родителем элемента с идентификатором *ROOT_ID*, а *tree* является таблицей, в которой хранится

дерево, тогда SQL-запрос на перемещение поддерева будет выглядеть следующим образом:

update tree set parent = ID where id = ROOT_ID;

Операция удаления поддерева.

Наиболее простым алгоритмом удаления поддерева из дерева, организованного методом прямых ссылок на родителей, является составление списка элементов поддерева и затем удаление элементов по данному списку. Пусть $C_1, C_2, \ldots, C_{n-1}, C_n$ есть список идентификаторов элементов удаляемого поддерева, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на удаление поддерева будет выглядеть следующим образом:

delete from tree where id in ($C_1, C_2, \ldots, C_{n-1}, C_n$)*;*

2. Метод материализацией пути

Материализованный путь рассматривается как некоторое строковое поле, которое состоит из строковых имен элементов, разделенных некоторым разделителем [47, 14, 33, 54]. В качестве разделителя обычно используется ‘.’ или ‘/’. Для того, чтобы уменьшить код в качестве строковых имен элементов, он используется нумерация братьев. Данный подход проиллюстрирован в таблице.

Таблица

Иллюстрация материализованного пути

Имя	**Путь**
King	1
Jones	1.1
Scott	1.1.1
Adams	1.1.1.1
Ford	1.1.2
Smith	1.1.2.1
Blake	1.2
Allen	1.2.1
Ward	1.2.2
Clark	1.3
Miller	1.3.1

В данной таблице путь 1.1.2 показывает, что элемент с именем Ford является вторым сыном элемента с именем Jones, а Jones, в свою очередь, является первым сыном элемента King.

Реляционное отношение, описывающее дерево в рамках модели материализации пути.

Реляционное отношение, описывающее дерево в рамках модели материализации пути, строится по следующим принципам:

– Каждый элемент иерархии кодируется уникальным идентификатором для связи с другими реляционными отношениями. Данный идентификатор занимает поле id.
– Каждый элемент иерархии кодируется в соответствии с методом материализации пути. Для кодирования элемента используются одно поле: path.
– Для каждого элемента хранится порядковый номер следующего его сына. Это делается для ускорения процесса добавления. Порядковый номер следующего сына хранится в поле: sibling_max.

Таблица

Реляционное отношение для метода материализации пути

Поле	**Тип**	**Описание**
Id	int4	Идентификатор элемента
Path	varchar(500)	Материализованный путь
Sibling_max	int4	Максимальный номер для сыновей

Операция добавления элемента в дерево.

Для добавления элемента в дерево организованное, по методу материализации пути, необходимо знать как идентификатор элемента сыном, которого будет новый элемент, так и количество сыновей данного элемента. Пусть *ID* есть идентификатор нового элемента, *PARENT_PATH* есть материализованный путь его родителя, *SIBLING_MAX* есть максимальный номер сына у родителя, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на вставку нового элемента будет выглядеть следующим образом:

insert into tree (id, path) values (ID, PARENT_PATH || '.' || (SIBLING_MAX+1));

Операция удаления элемента из дерева.

Как было сказано ранее, при удалении элемента из дерева необходимо не только удалить элемент, но и перенастроить связи оставшихся элементов для сохранения древовидной иерархии. Так, при организации дерева по методу материализации пути, предложенного Вадимом Тропашко, необходимо изменить значение материализованного пути у всех элементов поддерева, где удаляемый элемент являлся корнем. Пусть *PATH* есть материализованный путь удаляемого элемента, равный $C_1.C_2....C_{K-1}.C_K$, тогда материализованные пути его сыновей будут равны $C_1.C_2....C_{K-1}.C_K.C_{K+1,1}$, $C_1.C_2....C_{K-1}.C_K.C_{K+1,2}$,..., $C_1.C_2....C_{K-1}.C_K.C_{K+1,m-1}$, $C_1.C_2....C_{K-1}.C_K.C_{K+1,m}$, а материализованный путь его родителя равен $C_1.C_2....C_{K-1}$. Пусть *SIBLING_MAX* есть максимальный номер сына у родителя, тогда при удалении элемента с материализованным путем *PATH*, пути его сыновей должны иметь следующее значение: $C_1.C_2....C_{K-1}.C_K.(SIBLING\_MAX+1)$, $C_1.C_2....C_{K-1}.C_K.(SIBLING\_MAX+2)$,..., $C_1.C_2....C_{K-1}.C_K.(SIBLING\_MAX+m-1)$, $C_1.C_2....C_{K-1}.C_K.(SIBLING\_MAX+m)$. Если говорить об алгоритме решения данной задачи, то можно сказать, что необходимо переместить все поддеревья, корнями которых являются сыновья удаляемого элемента, так, чтобы родителем этих элементов стал родитель удаляемого элемента.

Операция поиска родителей элемента.

Для нахождения родителей определенного элемента дерева, при организации дерева по методу материализации пути, предложенного Вадимом Тропашко необходимо разложить материализованный путь элемента и получить пути для каждого из его родителей. Пусть материализованный путь для элемента равен $C_1.C_2....C_{K-1}.C_K$, тогда пути до его родителей будут равны: $\{C_1, C_1.C_2, \ldots, C_1.C_2 \ldots C_{K-1}, C_1.C_2...C_{K-1}.C_K\}$

Операция поиска сыновей элемента.

Нахождение сыновей элемента дерева, при его организации по методу материализации пути, требует выполнения одного SQL-запроса. Рассмотрим вариант SQL-запроса, который предлагает Joe Celko в работе «Hierarchical SQL» [12]. Пусть *PATH* есть материализованный путь до элемента, а *tree*

является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск сыновей элемента будет выглядеть следующим образом:

select id from tree where path like 'PATH.%' and path not like 'PATH.%.%' ;

Операция поиска элементов поддерева.

Нахождение элементов поддерева с определенным элементом в качестве корня является, в отличие от списка смежности, достаточно простой операцией на дереве организованном по методу материализации пути. Для решения задачи поиска элементов поддерева необходим один SQL-запрос. Пусть *PATH* есть материализованный путь до элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск элементов поддерева будет выглядеть следующим образом:

select id from tree where path like 'PATH.%';

Операция перемещения поддерева.

Перемещение поддерева внутри иерархии в случае, когда дерево организовано по методу материализации пути, требует выполнения достаточно сложного SQL-запроса, который затрагивает все элементы поддерева. Так как перемещение поддерева требует изменения значений материализованных путей у всех элементов поддерева. Пусть материализованный путь корня перемещаемого поддерева равен $C_1.C_2....C_{K-1}.C_K$, соответственно в множестве префиксов каждого из материализованных путей всех его потомков будет существовать префикс $C_1.C_2....C_{K-1}.C_K$. Пусть материализованный путь элемента, который будет новым родителем корня перемещаемого дерева равен, $A_1.A_2....A_{M-1}.A_M$. Тогда для перемещения выбранного поддерева необходимо заменить у всех элементов поддерева префикс $C_1.C_2....C_{K-1}.C_K$ на $A_1.A_2....A_{M-1}.A_M.A_{M+1}$, где A_{M+1} является новой частью пути, которая определяет положение корня переносимого поддерева среди сыновей нового корня. Для перемещения поддерева может быть предложен следующий алгоритм:

1: Определяем A_{M+1} = SIBLING_MAX+1, где SIBLING_MAX является максимальным индексом сыновей у нового корня перемещаемого дерева, материализованный путь которого равен $A_1.A_2....A_{M-1}.A_M$.

2: Определяем новый префикс для элементов поддерева равный $A_1.A_2....A_{M-1}.A_M.A_{M+1}$.

3: Перемещаем элементы поддерева. Изменяем старый префикс элементов поддерева равный $C_1.C_2....C_{K-1}.C_K$ на $A_1.A_2....A_{M-1}.A_M.A_{M+1}$. Данную операцию можно провести посредством выполнения следующего SQL-запроса:

update tree set path = overlay(path placing '$A_1.A_2....A_{M-1}.A_M.A_{M+1}$' from position('$C_1.C_2....C_{K-1}.C_K$' in path) for char_length('$C_1.C_2....C_{K-1}.C_K$')) where path='$C_1.C_2....C_{K-1}.C_K$' or path like '$C_1.C_2....C_{K-1}.C_K.\%$';

Операция удаления поддерева.

Удаление элементов поддерева с определенным элементом в качестве корня, как и поиск элементов поддерева, является достаточно простой операцией на дереве, организованном по методу материализации пути. Для решения задачи поиска элементов поддерева необходим один SQL-запрос. Пусть *PATH* есть материализованный путь до элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск элементов поддерева будет выглядеть следующим образом:

delete from tree where path='PATH' or path like 'PATH.%';

3. Метод вложенных множеств

Алгоритм кодирования вложенных множеств, который дал Nas Gaspiep [26], сложен для восприятия. Поэтому обычно пользуются алгоритмом, предложенным Joe Celko [13, 14, 12]. В данном алгоритме выполняется видоизмененный обход дерева в прямом порядке [98]. Суть изменения обхода сводится к тому, что обход для каждого поддерева выполняется не в два этапа, а в три: посетить корень; пройти все поддеревья, начиная с самого левого до самого правого; посетить корень. Формальное определение алгоритма кодирования вложенных множеств может выглядеть следующим образом:

Пусть i *есть индекс текущего элемента, который определяется линейным порядком, который задает измененный прямой обход дерева,* l_i, r_i *являются левым и правым индексами текущего элемента, а* n *есть количество элементов в дереве.*

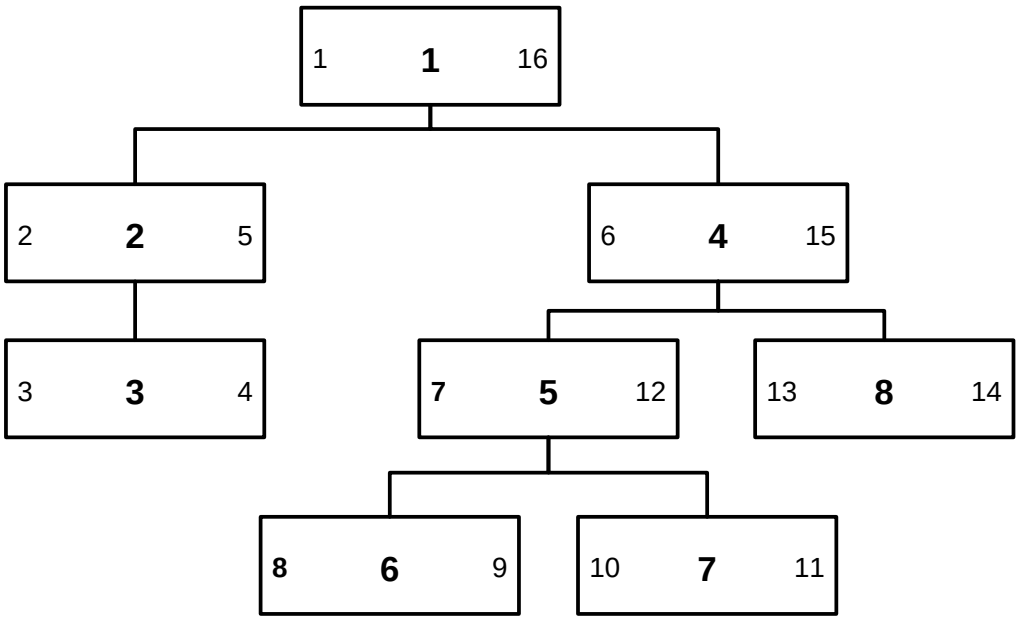


Рис. Иллюстрация алгоритма кодирования вложенных множеств

Алгоритм:

NS1: l = {0, 0, …, 0}; i=1; l_i=i

NS2: i = i+1; Если l_i = 0, то l_i=i иначе r_i=i

NS2: Если i > 2n, то выход из алгоритма иначе повторить NS2

Иллюстрация работы данного алгоритма приведена ранее на рисунке.

Реляционное отношение, описывающее дерево в рамках модели вложенных множеств.

Реляционное отношение, описывающее дерево в рамках модели вложенных множеств, строится по следующим принципам.

Каждый элемент иерархии кодируется уникальным идентификатором для связи с другими реляционными отношениями. Данный идентификатор занимает поле id.

Каждый элемент иерархии кодируется в соответствии с методом кодирования на основе прямого обхода дерева, описанного выше. Для кодирования элемента используются два поля: left_index и right_index – левый и правый индекс элемента иерархии соответственно.

Таблица

Пример таблицы для метода вложенных множеств

Поле	**Тип**	**Описание**
Id	int4	Идентификатор элемента
left_index	int4	Левый индекс узла
right_index	int4	Правый индекс узла

Операция добавления элемента в дерево.

Для добавления элемента в дерево, организованное по методу вложенных множеств, необходимо для нового элемента задать его левый и правый индексы, тем самым задать его местоположение в иерархии. Рассмотрим следующий алгоритм добавления элемента, как самого правого сына родителя [26]. Пусть *PARENT* есть идентификатор родителя, L_{PARENT} и

R_{PARENT}, его левый и правый индексы соответственно. Тогда о множестве левых индексов его сыновей $\{L_1, L_2, \ldots, L_m\}$ можно сказать, что $L_1=L_{PARENT}+1$, а множестве правых индексов его сыновей $\{R_1, R_2, \ldots, R_n\}$ можно сказать, что $R_N=R_{PARENT-1}$. Теперь мы можем сказать, что если добавить новый элемент в качестве самого правого сына, его левый индекс должен быть равен R_{PARENT}, а правый $R_{PARENT}+1$. Теперь, так как значениями индексов нового элемента стали индексы, использовавшиеся ранее для идентификации других элементов, то все индексы в правой части дерева, относительно нового элемента и прямого обхода, должны быть сдвинуты на два значения. То есть, если $L \geq R_{PARENT}$, то $L=L+2$ и если $R \geq R_{PARENT}$, то $R=R+2$, где L,R левый и правый индексы элементов дерева соответственно.

При использовании описанного алгоритма SQL-запрос на вставку нового элемента будет выглядеть следующим образом:

select R_{PARENT} from nested_sets where id =X;

update nested_sets set right_index=right_index+2 where right_index >= R_{PARENT};

update nested_sets set left_index=left_index+2 where left_index >= R_{PARENT};

insert into nested_sets values (NEW_ID, R_{PARENT}, R_{PARENT}+1);

Операция удаления элемента из дерева.

В том случае, если необходимо поддерживать дерево, чтобы оно отвечало приведенному определению вложенных множеств, то для удаления элемента из дерева необходимо не только удалить выбранный элемент, но и перестроить левый и правый индексы у оставшихся элементов так, чтобы не нарушилась иерархия. Для этого необходимо у всех потомков данного элемента уменьшить индексы на единицу, а у правой части дерева, относительно удаляемого элемента, на два. Пусть X есть идентификатор элемента, который необходимо удалить, L_X и R_X, его левый и правый индексы соответственно. Тогда при удалении элемента необходимо следующим образом изменить индексы всего дерева:

1. Если $L_i \geq R_X$, то $L_i=L_i-2$, где L_i левый индекс i-го элемента дерева;
2. Если $R_i \geq R_X$, то $R_i=R_i-2$, где R_i правый индекс i-го элемента дерева;
3. Если $L_i>L_X$ и $L_i<R_X$, то $L_i=L_i-1$ и $R_i=R_i-1$, где L_i левый, а R_i правый индекс i-го элемента дерева;

При использовании описанного алгоритма SQL-запрос на удаление элемента будет выглядеть следующим образом:

select L_X, R_X from nested_sets where id =X;

update nested_sets set right_index=right_index-2 where right_index > R_X;

update nested_sets set left_index=left_index-2 where left_index > R_X;

update nested_sets set left_index=left_index-1, right_index=right_index-1 where left_index > L_X and left_index < R_X;

delete from nested_sets where id = X;

Однако для работы с иерархией необязательно всегда поддерживать дерево отвечающее определению . Даже при удалении элемента без перестройки дерева, такое дерево не теряет основного свойства дерева, построенного по методу вложенных множеств. В таком случае при удалении элемента SQL-запрос на удаление элемента будет выглядеть следующим образом:

delete from nested_sets where id = X;

Операция поиска родителей элемента.

Для поиска родителей элемента в дереве, организованном по методу вложенных множеств, достаточно выполнения одного SQL-запроса. Использование левых и правых индексов в качестве границ множеств позволяет организовать единственный запрос. Пусть *X* есть идентификатор элемента, а *netsted_sets* является таблицей, в которой хранится дерево. Тогда SQL-запрос на поиск родителей элемента будет выглядеть следующим образом:

```
select ns1.id
from
   nested_sets ns1, nested_sets ns2
```

where

ns2.id = X and

ns2.left_index > ns1.left_index and

ns2.left_index < ns1.right_index;

Операция поиска сыновей элемента.

В дереве, построенном по методу вложенных множеств, нет прямых указаний на сыновей элементов. Однако существует метод определения сыновей по уровню дерева, на котором находится элемент. Из определения дерева можно вывести следующее определение сыновей: *сыновьями элемента X находящегося на уровне дерева l являются элементы, которые являются его потомками и находятся на l+1 уровне дерева*.

Однако в модели вложенных множеств нет прямого указания на уровень элемента, на котором он находится в дереве. Для того, чтобы определить уровень элемента в дереве, Celko предлагает подсчитывать количество родителей элемента [14].

Пусть *X* есть идентификатор элемента, а *netsted_sets* является таблицей, в которой хранится дерево. Тогда SQL-запрос на подсчет родителей элемента будет выглядеть следующим образом:

select count(ns1.id) as count

from

nested_sets ns1, nested_sets ns2

where

ns2.id = 10 and

ns2.left_index > ns1.left_index and

ns2.left_index < ns1.right_index;

Для нахождения уровней всех элементов, которые являются элементами поддерева, корнем которого является элемент *X* SQL-запрос будет выглядеть так:

select ns2.id as id, count(ns1.id) as count

from nested_sets ns1, nested_sets ns2, nested_sets ns3

where

ns2.left_index > ns1.left_index and

ns2.left_index < ns1.right_index and

ns2.left_index > ns3.left_index and

ns2.left_index < ns3.right_index and

ns3.id = X.id

group by ns2.id

Теперь для нахождения сыновей необходимо из всех элементов поддерева оставить только те, которые находятся на следующем уровне дерева, чем рассматриваемый элемент *X*. Алгоритм нахождения элементов поддерева будет рассмотрен ниже. Пусть $Level_X$ есть уровень, на котором находится элемент *X*. Тогда объединенный SQL-запрос для поиска сыновей элемента дерева, организованного по методу вложенных множеств, выглядит следующим образом:

select id from (

select ns2.id as id, count(ns1.id) as count

from nested_sets ns1, nested_sets ns2, nested_sets ns3

where

ns2.left_index > ns1.left_index and

ns2.left_index < ns1.right_index and

ns2.left_index > ns3.left_index and

ns2.left_index < ns3.right_index and

ns3.id = 62

group by ns2.id

) as levels where count = $Level_X$*;*

Операция поиска элементов поддерева.

SQL-запрос поиска элементов поддерева определенного элемента в дереве, организованном по методу вложенных множеств, является симметричным по отношению к алгоритму поиска родителей элемента. Пусть *X* есть идентификатор элемента, а *netsted_sets* является таблицей, в

которой хранится дерево. Тогда SQL-запрос на поиск элементов поддерева определенного элемента будет выглядеть следующим образом:

```
select ns2.id
from  nested_sets ns1, nested_sets ns2
where
        ns1.id = X.id and
        ns2.left_index > ns1.left_index and
        ns2.left_index < ns1.right_index;
```

Операция перемещения поддерева.

Перемещение поддерева при организации древовидной иерархии по методу вложенных множеств представляет собой изменение кодов элементов поддерева.

Nas Gaspiep предлагает следующий алгоритм изменения кодов [26]:

1. Вычислить количество элементов в поддереве.
2. Умножить все левые и правые индексы элементов поддерева на -1.
3. Преобразовать дерево, предположив, что поддерево было удалено.
4. Увеличить левые индексы элементов, которые больше чем правый индекс элемента, куда будет перемещено дерево, на (2*количество элементов в поддереве+2).
5. Увеличить правые индексы элементов, которые больше или равны правому индексу элемента, куда будет перемещено дерево, на (2*количество элементов в поддереве+2).
6. Найти наибольший отрицательный левый индекс в дереве, умножить на -1, и результат назвать *x*.
7. Найти левый индекс элемента, куда перемещается поддерево, и назвать его *y*.
8. Вычислить *z=(y-x+1)*. Вычесть из всех отрицательных левых и правых индексов *z*.
9. Умножить все отрицательные индексы на -1.

Как уже было сказано выше, можно отказаться от преобразования дерева после удаления из него элемента или поддерева. Поэтому можно пренебречь преобразованием. Однако в таком случае пункты 4, 5 не будут выполнять свои функции. Может произойти ситуация, когда $(r_{max} - l_{min}) / 2 > C$, где r_{max}, l_{min} – наибольший правый индекс и наименьший левый индекс рассматриваемого дерева соответственно, а C – количество элементов поддерева. Такая ситуация может произойти, если из данного поддерева до перемещения был удален элемент или поддерево. Чтобы решить возникшую проблему в пунктах 4 и 5 в качестве множителя следует использовать не $2*C+2$, а $r_{max} – l_{min} +2$.

Операция удаления поддерева.

Удаление поддерева в дереве, организованном по методу вложенных множеств, в чем-то схоже с удалением одного элемента. Поэтому в том случае, если необходимо поддерживать дерево, чтобы оно отвечало приведенному определению вложенных множеств (см. опр. 1.1.), то для удаления поддерева необходимо не только удалить выбранное поддерево, но и перестроить левые и правые индексы у оставшихся элементов так, чтобы не нарушилась иерархия. Для этого необходимо у правой части дерева относительно удаляемого элемента уменьшить левые и правые индексы. Пусть X есть идентификатор корня поддерева, которое необходимо удалить, L_X и R_X, его левый и правый индексы, соответственно, C есть количество элементов в поддереве. Тогда при удалении поддерева необходимо следующим образом изменить индексы всего дерева:

1. Если $L_i > R_X$, то $L_i=L_i-2*C$, где L_i левый индекс i-го элемента дерева;
2. Если $R_i > R_X$, то $R_i=R_i-2*C$, где R_i правый индекс i-го элемента дерева.

При использования описанного алгоритма SQL-запрос на удаление элемента будет выглядеть следующим образом:

select L_X, R_X from nested_sets where id =X;

select count(ns1.id) as C

```
from
   nested_sets ns1, nested_sets ns2
where
    ns2.id = X and
    ns2.left_index > ns1.left_index and
    ns2.left_index < ns1.right_index;
```

*update nested_sets set right_index=right_index-2*C where right_index > R_X;*

*update nested_sets set left_index=left_index-2*C where left_index > R_X;*

```
delete from nested_sets where
 id in (
        select count(ns1.id) as C
        from
                nested_sets ns1, nested_sets ns2
        where
                ns2.id = X and
                ns2.left_index > ns1.left_index and
                ns2.left_index < ns1.right_index;
);
```

Однако для работы с иерархией необязательно поддерживать дерево, отвечающее определению 1.1. Даже при удалении поддерева без перестройки дерева такое дерево не теряет основного свойства дерева, построенного по методу вложенных множеств (см. опр. 1.2). В таком случае при удалении поддерева SQL-запрос будет выглядеть следующим образом:

```
delete from nested_sets where
 id in (
        select count(ns1.id) as C
        from
                nested_sets ns1, nested_sets ns2
        where
```

ns2.id = X and

ns2.left_index > ns1.left_index and

ns2.left_index < ns1.right_index;

);

Алгоритм преобразования древовидной иерархии в реляционную модель вложенных множеств.

Как пишет Joe Celko [14], и показывает практика время выполнения операции добавления элементов в рамках модели вложенных множеств с ростом объема иерархии. Это происходит вследствие необходимости изменения кодов значительной части дерева при добавлении каждого элемента. Для более быстрого построения иерархии в реляционной модели вложенных множеств производят трансформацию уже заполненной иерархии в рассматриваемую модель. Joe Celko описывает стековый алгоритм трансформации [14]. Однако можно предложить более простой в кодировании алгоритм. Основу алгоритма составляет следующее выражение.

Пусть

l_i – левый индекс *i*-го элемента,

r_i – правый индекс *i*-го элемента,

d_i – количество потомков *i*-го элемента, тогда

$$r_i - l_i = 2*(d_i)+1$$

Алгоритм может быть описан, представленной на следующем листинге, рекурсивной функцией.

Листинг

Рекурсивная функция преобразования в модель вложенных множеств

```
(1) Функция ПостроитьВложенныеМножестваИз(Дерево, Текущий
        элемент, Левый Индекс)
    {
(2)   Если Текущий элемент является корнем, то Левый индекс = 1;
(3)   Текущий индекс = Левый индекс + 1;
(4)   Для всех детей текущего элемента в цикле вызвать функцию
      {
(5)     Текущий индекс = ПостроитьВложенныеМножестваИз(Дерево,
                Ребенок, Текущий индекс)
```

```
      }
(6)   Добавить в реляционное отношение вложенных множеств
текущий
      элемент, с левым индексом равным переменной Левый индекс,
      а правым равным переменной Текущий индекс;
(7)   Вернуть из функции Текущий индекс +1;
    }
```

Для преобразования любой древовидной иерархии в модель вложенных множеств необходимо выполнить приведенный алгоритм, начиная с корня иерархии.

4. Метод вложенных интервалов

В качестве кодов элементов дерева используются рациональные числа *a/b*, где *a≥b≥1* и *НОД(a,b)=1*. Данные числа получены на основе материализованного пути с помощью конечной цепной дроби $[q_1, q_2, \ldots, q_n]$, где $q_1, q_2, \ldots, q_n$ представляют собой шаги материализованного пути в дереве. При материализации пути в данном случае каждый шаг идентифицируется порядковым номер элемента среди своих братьев. Так, например, для материализованного пути *3.12.5.1.21* код строится из следующей цепной дроби:

$$3+\cfrac{1}{12+\cfrac{1}{5+\cfrac{1}{1+\cfrac{1}{21}}}}=\frac{4913}{1594}$$

Соответственно, для преобразования материализованного пути в рациональное выражение можно воспользоваться *законом подходящих дробей*, а для преобразования рационального выражения в материализованный путь следует воспользоваться алгоритмом последовательного выделения целой части числа, который в случае рационального числа совпадает с алгоритмом Эвклида для нахождения НОД. [71].

Также проводится соответствие рациональных чисел, получаемых при помощи цепных дробей с вложенными интервалами. Для того, чтобы найти границы интервала, необходимо преобразовать конечную цепную дробь в функцию и привести ее к виду: $w=f(z)=\frac{az+b}{cz+d}$, где $ad-bc\neq 0$, то есть привести к форме Мёбиуса [38]. Также существуют параллели между представлением функции в форме Мёбиуса и алгеброй матриц 2x2, что дает мощный аппарат для преобразований кодов [52].

Тем самым, определяется следующий аппарат преобразований:

– Рациональное число → Материализованный путь (Алгоритм Евклида).

– Материализованный путь → Рациональное число (упрощение цепной дроби).
– Форма Мёбиуса *(ax+b)/(cx+d)* ↔ Интервал *(a/c, (a+b)/(c+d)]*.
– Интервал *(a/c, (a+b)/(c+d)]* → Рациональное число *a/c*.
– Материализованный путь → Форма Мёбиуса (подстановка *1/x* в цепную дробь и упрощение).

Алгоритм преобразования рационального числа в материализованный путь.

Так как материализованный путь в рассматриваемом методе кодируется с помощью цепных дробей для преобразования рационального числа в материализованный путь, необходимо преобразовать число в конечную цепную дробь. Воспользуемся для этого алгоритмом, приведенным в методическом пособии А.Т. Гайнова «Теория чисел. Часть 1.» [71].

Данный алгоритм совпадает с алгоритмом Евклида для нахождения НОД чисел $a,b \geq 1$. Пусть $\alpha = \frac{a}{b}; a,b \in Z$, то α разлагается в конечную целую цепную дробь $[q_1, q_2, \ldots, q_n]$, где $q_1, q_2, \ldots, q_n$ представляют собой шаги материализованного пути в дереве.

Шаг 1. $a = bq_1 + r_1, 0 \leq r_1 < b \Rightarrow \frac{a}{b} = q_1 + \frac{r_1}{b}$. Отсюда видно, что $q_1 = [\frac{a}{b}], \frac{r_1}{b} = \{\frac{a}{b}\}$. Если $r_1 = 0$, то $\frac{a}{b} = [q_1]$.

Шаг 2. Если $r_1 > 0$, тогда $\frac{b}{r_1} > 1$. Отсюда

$$b = r_1 q_2 + r_2, 0 \leq r_2 < r_1 \Rightarrow \frac{b}{r_1} = q_2 + \frac{r_2}{r_1}$$

$$q_2 = [\frac{b}{r_1}], \frac{r_2}{r_1} = \{\frac{b}{r_1}\} \text{ и т.д.}$$

Шаг n-1. $r_{n-3}=r_{n-2}q_{n-1}+r_{n-1}$, $r_{n-2}=r_{n-1}q_n$, тогда r_n – НОД чисел a,b, а материализованный путь будет иметь вид $q_1, q_2, \ldots, q_n$ [71].

Алгоритм преобразования материализованного пути в рациональное число

Так как материализованный путь в рассматриваемом методе кодируется с помощью цепных дробей, для преобразования материализованного пути в рациональное число необходимо преобразовать связанную с ним конечную цепную дробь в число. Воспользуемся для этого законом подходящих дробей [71].

Пусть $q_1, q_2, \ldots, q_n$ есть материализованный путь определенного элемента дерева, тогда $\alpha = [q_1, q_2, \ldots, q_n]$ является определяемой данным материализованным путем целой цепной дробью, а $q_1, q_2, \ldots, q_n$ являются ее независимыми переменными. Тогда ее подходящие дроби $\frac{P_k}{Q_k}, k = 0,1,\ldots,n-1$ удовлетворяют следующему рекуррентному правилу, называемому *законом подходящих дробей*:

$$\begin{aligned} &P_0 = 1, Q_0 = 0, P_1 = q_1, Q_1 = 1, \\ &P_{k+1} = q_{k+1}P_k + P_{k-1}, Q_{k+1} = q_{k+1}Q_k + Q_{k-1}, \forall k \geq 1. \end{aligned} \qquad (П4.1)$$

Проблема пересечения интервалов.

В ходе анализа функций цепных дробей было выявлено, что возможны случаи, когда происходит касание интервалов кодирования. Рассмотрим два элемента с кодами 2.3.4 и 2.3.3.1. Цепная дробь для первого элемента будет выглядеть, как $2 + \frac{1}{3 + \frac{1}{4}}$, а для второго $2 + \frac{1}{3 + \frac{1}{3 + \frac{1}{1}}}$. Не трудно заметить, что они равны. Определение, описывающее данный случай, может быть следующим:

Интервалы двух элементов будут касаться, если будут выполняться следующие условия:

1. $x_1, x_2, \ldots, x_{n-1}, x_n$ *материализованный путь первого элемента;*
2. $x_1, x_2, \ldots, x_{n-1}, y, z$ *материализованный путь второго элемента;*

3. $z=1;$

4. $y = x_{n-1}$

5. $n \geq 1.$

Решить проблему пересечения интервалов можно, отказавшись при кодировании от индекса со значением «1».

Проблема скорости роста числителя и знаменателя дроби.

Следует заметить, что при кодировании вложенных интервалов с помощью цепных дробей существует проблема скорости роста значений числителя и знаменателя дроби. Попробуем численно описать данную проблему и определить на сколько она существенна. Числитель и знаменатель вычисляются по схожим формулам, однако числитель всегда больше. Соответственно, можно ограничиться анализом поведения числителя.

$$P_0 = 1, P_1 = q_1,$$
$$P_{k+1} = q_{k+1}P_k + P_{k-1}, \forall k \geq 1. \qquad (П4.2)$$

Далее докажем две гипотезы:

1. С увеличением количества братьев возрастает числитель, который имеет брат с большим порядковым номером;

2. С увеличением количества уровней в пути от элемента до корня значение числителя возрастает.

По первой гипотезе следует сказать, что количество братьев прямо влияет на кодирование материализованного пути, так как порядковый номер элемента среди братьев участвует в кодировании последнего шага пути до элемента. Тем самым можно отметить, что при увеличении количества братьев увеличится и максимальный порядковый номер. Пусть элемент E имеет максимальный порядковый номер среди своих братьев. Рассмотрим два случая. Первый, пусть элемент E имеет порядковый номер m. Второй, пусть тот же элемент E имеет порядковый номер n, что $n > m$. Исходя из гипотезы и формулы вычисления числителя П4.1 для цепной дроби, нам необходимо доказать, что $nP_k+P_{k-1} > mP_k+P_{k-1}$. Отняв из обеих частей

неравенства P_{k-1}, и затем разделив обе части неравенства на P_k, мы получим неравенство $n > m$, которое по условию истинно, следовательно, и выдвинутая нами гипотеза верна.

Доказательство второй гипотезы приводит нас к тому, что если у какого-либо элемента дерева есть сын, то числитель сына больше числителя родителя. Пусть существует некоторый элемент дерева E, который находится на k-ом уровне. Тогда по формуле П4.2 его числитель равен $P_k = q_k P_{k-1} + P_{k-2}$. Тогда если у элемента Е есть сыновья, то числитель первого из них будет равен $P_{[k+1,1]} = 1*(P_{k-1}+P_{k-1})+P_{k-1}$. Из полученных формул видно, что числитель уже первого сына элемента E больше числителя элемента E. Исходя же из доказанной первой гипотезы, можно утверждать, что все следующие сыновья элемента E будут иметь еще больший числитель. Тем самым, выдвинутая нами вторая гипотеза доказана.

Теперь попробуем дать численную оценку скорости роста числителя рационального числа, кодирующего элементы дерева. Для простоты оценки попробуем видоизменить формулу 2.4. Во-первых, перейдем от формулы с бесконечным числом параметров к формуле с двумя параметрами. Данная формула имеет бесконечно изменяемое множество параметров оно равно $\{k, q_1, q_2, \ldots, q_{k-1}, q_k\}$, где $k \geq 1$. Мы же условимся, что при кодировании элемента он сам и все его родители имеют один и тот же порядковый номер среди братьев, равный q. Такое условие дает нам более простую формулу:

$$\begin{aligned} &P_0 = 1, P_1 = q, \\ &P_{k+1} = qP_k + P_{k-1}, \forall k \geq 1. \end{aligned} \qquad (П4.3)$$

Более простая формула П4.3, являясь рекурсивной, трудно поддается анализу в силу большой трудоемкости вычисления ее значений. Попробуем заменить функцию P из формулы П4.3 ее нижней оценкой - некоторой функцией Q, что $P_k \geq Q_k$, $k \geq 1$, и Q будет более простой функцией от двух переменных.

Пусть $Q = q^n$. Теперь докажем верность неравенства $P_k \geq Q_k$, $k \geq 1$. Произведем доказательство по методу математической индукции.

1. $k = 1, P_1 = q, Q_1 = 1, P_1 \geq Q_1;$
2. $k = 2, P_1 = q^2 + 1, Q_1 = q^2, P_1 \geq Q_1;$
3. *Посылка индукции:* $P_k \geq Q_k$ *и* $P_{k+1} \geq Q_{k+1}$
4. *Докажем, что* $P_{k+2} \geq Q_{k+2}$
5. $P_{k+2} = qP_{k+1} + P_k, Q_{k+2} = qQ_{k+1}$
6. *Исходя из посылки индукции и того, что* $Pk > 0$ *при* $k \geq 0$*, можно утверждать, что неравенство* $qP_{k+1} + P_k \geq qQ_{k+1}$ *верно. Следовательно, неравенство* $P_k \geq Q_k$ *верно при* $k \geq 1$.

Соответственно, функция Q является нижней оценкой P

Современные системы управления базами данных для представления целых чисел имеют встроенные типы данных. На данный момент существуют типы данных для целых чисел, размер которых составляет 8 байт. Тем самым данные типы данных могут закодировать 2^{64} значений. Попробуем оценить численные параметры элемента числитель, который равен данному пределу. Соответственно, получим:

$$Q = q^n = 2^{64} \Rightarrow q = e^{\frac{\ln 2^{64}}{n}}, n = \frac{\ln 2^{64}}{\ln q} \qquad (П4.4)$$

Исходя из формул П4.4, и проведенных вычислений (см. рис. ниже) можно сделать следующие утверждения:

- $q = 2 \Rightarrow n = 64;$
- $q = 10 \Rightarrow n \approx 19;$

А так как $P \geq Q$ то:

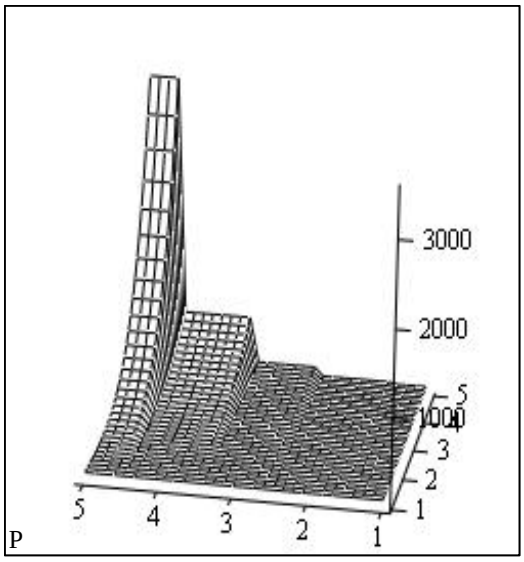
- $q = 2, n = 64 \Rightarrow P(q,n) \geq 2^{64};$
- $q = 10, n = 19 \Rightarrow P(q,n) \geq 2^{64};$

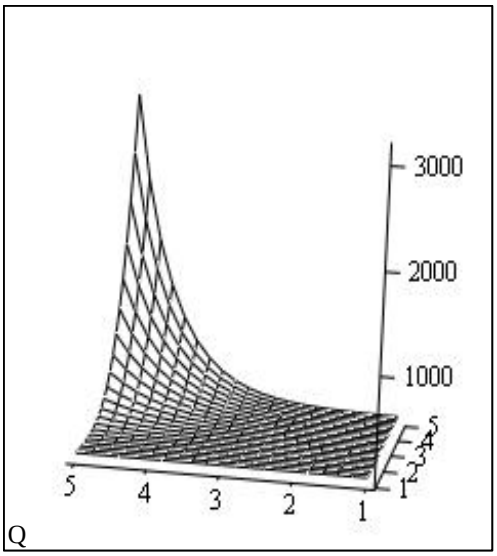
Вместе с тем, предел восьмибайтового целочисленного типа данных будет достигнут или превышен, при условии существования элемента на 64-м уровне и все его родители были братьями со вторым порядковым номером. Или же данный предел будет достигнут или превышен, при условии существования элемента на 19-м уровне и все его родители были братьями с десятым порядковым номером. Тем самым, мы показали, что возможны

ситуации, что уже при количестве элементов в дереве равном 128 элементам будет превышен восьмибайтовый предел кодирования числителя. Можно сделать выводы, что необходимо аккуратно применять данный метод для различных типов деревьев. Данный метод устанавливает ограничения на структуру дерева.

$$P(q,k) := \begin{cases} 0 & \text{if } \text{round}(k,0) = 0 \\ q & \text{if } \text{round}(k,0) = 1 \\ q \cdot P(q,k-1) + P(q,k-2) & \text{otherwise} \end{cases}$$

$$Q(q,k) := q^k$$





$n := 1..100$

$$q(n) := \exp\left(\frac{\ln\left(2^{64}\right)}{n}\right)$$

$q(n) =$

$1.845 \cdot 10^{19}$
$4.295 \cdot 10^{9}$
$2.642 \cdot 10^{6}$
$6.554 \cdot 10^{4}$
$7.132 \cdot 10^{3}$
$1.625 \cdot 10^{3}$
565.294
256
138.248
84.449
56.422
40.317
30.338
23.776
19.248
16

$$n(q) := \frac{\ln\left(2^{64}\right)}{\ln(q)}$$

$q := 2..100$

$n(q) =$

64
40.38
32
27.563
24.759
22.797
21.333
20.19
19.266
18.5
17.852
17.295
16.81
16.381
16
15.658

Рис. Анализ функции вычисления числителя цепной дроби, кодирующей элемент дерева в модели вложенных интервалов

Когда, ввиду структуры дерева и количества элементов, возможен выход за приделы встроенных целочисленных типов, следует использовать для хранения рационального числа другие типы, позволяющие хранить бинарные данные с возможностью их индексирования. Также следует обратить внимание на то, что программная часть должна позволять производить операции с большими целыми числами.

Реляционное отношение, описывающее дерево в рамках модели вложенных интервалов.

Реляционное отношение, описывающее дерево в рамках модели вложенных интервалов, строится по следующим принципам:

- Каждый элемент иерархии кодируется уникальным идентификатором для связи с другими реляционными отношениями. Данный идентификатор занимает поле id.
- Каждый элемент иерархии кодируется в соответствии с методом кодирования на основе цепных дробей, рассмотренных выше. Для кодирования элемента используются два поля: numerator и denominator – числитель и знаменатель кодирующей дроби, соответственно.

Таблица

Реляционное отношение для метода вложенных интервалов

Поле	**Тип**	**Описание**
id	int8	Идентификатор элемента
numerator	int8	Числитель рационального числа
denominator	int8	Знаменатель рационального числа
last_child_numerator	int8	Числитель рационального числа последнего ребенка текущего элемента
last_child_denominator	int8	Знаменатель рационального числа последнего ребенка текущего элемента

Для каждого элемента хранится код следующего его сына. Это делается для ускорения процесса добавления. Код следующего сына также

хранится в виде рационального кода, для которого использованы следующие поля: last_child_numerator и last_child_denominator.

Операция добавления элемента в дерево.

Для добавления нового элемента в дерево, организованное по методу вложенных интервалов, кодирующихся цепными дробями, в начале необходимо найти код нового элемента. Для этого предлагается следующий алгоритм:

1. Преобразуем рациональное число, кодирующее родителя в его материализованный путь *M*.
2. Находим материализованный путь самого правого сына родителя и определяем *K* как идентификатор последнего шага его пути + 1.
3. Если у родителя нет сыновей, то определяем *K=1*.
4. Составляем материализованный путь для нового элемента M_{new} путем конкатенации *M* и *K*.
5. Преобразуем M_{new} в рациональное число и получаем код нового элемента.

Можно предложить другой вариант алгоритма добавления элемента в дерево, организованное по методу вложенных интервалов, кодирующихся цепными дробями. Такой алгоритм предлагает несколько иной способ кодирования шагов пути. В виду того, что поиск сыновей является достаточно сложной задачей, следует кодировать элементы дерева таким образом, что необходимость в поиске сыновей, при определении шага пути, исчезнет. Можно определять шаг пути как код идентификатора элемента.

1. Преобразуем рациональное число, кодирующее родителя в его материализованный путь *M*.
2. Определяем шаг пути *K* как код идентификатора элемента.
3. Составляем материализованный путь для нового элемента M_{new} путем конкатенации *M* и *K*.

4. Преобразуем M_{new} в рациональное число и получаем код нового элемента.

У данного алгоритма есть недостаток: большая, чем у предыдущего алгоритма, скорость роста значений числителя и знаменателя дроби.

Существует еще возможность расширения кортежа, описывающего элемент дерева двумя полями, которые бы хранили числитель и знаменатель последнего сына рассматриваемого элемента. В данном случае вычисление значения пути для следующего сына, а затем преобразование его в дробь не составит труда. Алгоритм добавления нового элемента может быть следующим:

1. Преобразуем рациональное число, кодирующее последнего сына в его материализованный путь M.

2. Увеличиваем значение последнего шага пути на единицу – теперь M является материализованным путем для нового элемента.

3. Преобразуем M в рациональное число, и получаем код нового элемента.

Операция удаления элемента из дерева.

Для удаления элемента из дерева, построенного по методу вложенных интервалов, которые кодируются цепными дробями, необходимо воспользоваться следующим алгоритмом:

Шаг 1. Найти родителя данного элемента. Обозначим родительский элемент P.

Шаг 2. Найти всех сыновей удаляемого элемента. Пусть множество D есть множество сыновей.

Шаг 3. Перенести все поддеревья, корнями которых являются элементы из D в P.

Шаг 4. Удалить рассматриваемый элемент.

Алгоритмы поиска сыновей и перемещения поддеревьев приведены ниже.

Операция поиска родителей элемента.

Поиск родителей элемента в дереве, построенном по методу вложенных интервалов, которые кодируются цепными дробями, необходимо преобразовать рациональное число в материализованный путь, который и определит родителей элемента.

Пусть после преобразования материализованный путь рассматриваемого элемента имеет вид $q_1, q_2, \ldots, q_n$, тогда родители рассматриваемого элемента будут иметь следующие материализованные пути $\{q_1; q_1, q_2; \ldots; q_1, q_2, \ldots, q_n\}$.

Операция поиска сыновей элемента.

Для того, чтобы найти сыновей определенного элемента в дереве, организованном по методу вложенных интервалов, кодирующихся цепными дробями, необходимо воспользоваться следующим алгоритмом:

Шаг 1. Определить k как количество шагов в материализованном пути рассматриваемого элемента.

Шаг 2. Найти все элементы поддерева, в котором рассматриваемый элемент является корнем.

Шаг 3. Выбрать среди элементов поддерева те, у которых количество шагов в материализованном пути равно $k+1$.

Операция поиска элементов поддерева.

Для того, чтобы в дереве, организованном по методу вложенных интервалов кодирующихся цепными дробями, найти элементы поддерева, в котором определенный элемент является корнем, необходимо сначала по рациональному коду рассматриваемого элемента найти определяемый им интервал. Для этого нужно воспользоваться следующим алгоритмом:

Шаг 1. Пусть α является рациональным кодом рассматриваемого элемента, что $\alpha = \frac{a}{b}; a, b \in Z$.

Шаг 2. Определить материализованный путь родителя. Перейти от рационального кода элемента к материализованному пути. Пусть $q_1, q_2, \ldots, q_{n-1}, q_n$ является материализованным путем рассматриваемого элемента.

Шаг 3. Определить материализованный путь родителя. Если q_1, q_2, ..., q_{n-1}, q_n является материализованным путем рассматриваемого элемента, тогда q_1, q_2, ..., q_{n-1} является материализованным путем его родителя.

Шаг 4. Перейти от материализованного пути родителя к его рациональному коду. Пусть β является рациональным кодом родителя, что $\beta = \frac{c}{d}; c, d \in Z$.

Шаг 5. Построить уравнение, определяющее интервал рассматриваемого элемента. Данное уравнение в форме Мёбиуса имеет вид: $\frac{ax+c}{bx+d}$, и определяет следующий интервал: $\left[\frac{a+c}{b+d}, \frac{a}{b}\right)$ [52].

Как было сказано выше, при кодировании нельзя использовать число 1 в качестве индекса в пути элементов дерева. Исходя из данного утверждения, можно представить следующий интервал, в котором будут находиться потомки элемента: $\left(\frac{a+c}{b+d}, \frac{a}{b}\right)$. Также следует обратить внимание на то, что уравнение формы Мёбиуса может определять как $\left(\frac{a+c}{b+d}, \frac{a}{b}\right)$, так и $\left(\frac{a}{b}, \frac{a+c}{b+d}\right)$ интервалы. Данный факт не описан в работе Tropashko, однако несомненно заслуживает внимания. Например, кодируем элемент $2.2.2 \Rightarrow 2 + \frac{1}{2+\frac{1}{2}} = \frac{12}{5}$, кодируем родителя $2.2 \Rightarrow 2 + \frac{1}{2} = \frac{5}{2}$. Следовательно, интервал для потомков элемента 2.2.2 по форме Мёбиуса будет равен $\left(\frac{12}{5}, \frac{12+5}{5+2}\right) = \left(\frac{12}{5}, \frac{17}{7}\right) = (2.4, \sim 2.42)$. Однако при кодировании элемента $2.2 \Rightarrow 2 + \frac{1}{2} = \frac{5}{2}$, кодируем родителя $2 \Rightarrow 2$. Следовательно, интервал для потомков элемента 2.2 по форме Мёбиуса равен $\left(\frac{5+2}{2+1}, \frac{5}{2}\right) = \left(\frac{7}{3}, \frac{5}{2}\right) = (2.(3), 2.5)$.

Шаг 6. При использовании описанного алгоритма SQL-запрос на поиск элементов определенного поддерева будет выглядеть следующим образом:

select id from nested_intervals where numerator/denominator < max(a/b, (a+c)/(b+d)) and numerator/denominator > min(a/b, (a+c)/(b+d));

Операция перемещения поддерева.

Перемещение поддерева при организации древовидной иерархии по методу вложенных интервалов, которые кодируются цепными дробями, представляет собой изменение кодов элементов рассматриваемого поддерева.

Пусть $\alpha=[a_1, a_2, \ldots,a_n]$ и $\beta=[b_1, b_2, \ldots,b_m]$ являются конечными целыми цепными дробями, которые определяют корень перемещаемого поддерева, и тот элемент, который будет новым родителем рассматриваемого поддерева, соответственно. Для начала попытаемся переместить корень поддерева к новому родителю. Для того, чтобы это сделать, необходимо найти свободный интервал в интервале, определяемом родителем. Для этого можно воспользоваться способом поиска свободного интервала из алгоритма добавления нового элемента в дерево описанного выше. Тогда после перемещения цепная дробь, определяющая корень поддерева, будет иметь вид $\alpha=[b_1, b_2, \ldots,b_m, x]$, где x есть новый идентификатор элемента на уровне, а материализованный путь будет выглядеть как $b_1, b_2, \ldots,b_m, x$. Теперь, чтобы переместить все остальные элементы, необходимо изменить их коды. В терминах материализованного пути это будет выглядеть следующим образом. Множество материализованных путей элементов рассматриваемого поддерева до перемещения выглядит следующим образом: $\{a_1, a_2, \ldots,a_n, q_{(1,1)}, q_{(1,2)}, \ldots q_{(1,n1)}; a_1, a_2, \ldots,a_n, q_{(2,1)}, q_{(2,2)}, \ldots q_{(2,n1)}; \ldots; a_1, a_2, \ldots,a_n, q_{(t,1)}, q_{(t,2)}, \ldots q_{(t,nt)}\}$. Можно заметить, что часть $a_1, a_2, \ldots,a_n$ во всех материализованных путях является общей. Для того, чтобы переместить элементы поддерева на новое место, необходимо заменить существующую общую часть на новую - $b_1, b_2, \ldots,b_m, x$. Результирующее множество материализованных путей элементов рассматриваемого поддерева будет выглядеть следующим образом: $\{b_1, b_2, \ldots,b_m, x, q_{(1,1)}, q_{(1,2)}, \ldots q_{(1,n1)}; b_1, b_2, \ldots,b_m, x, q_{(2,1)}, q_{(2,2)}, \ldots q_{(2,n1)}; \ldots; b_1, b_2, \ldots,b_m, x, q_{(t,1)}, q_{(t,2)}, \ldots q_{(t,nt)}\}$.

Операция удаления поддерева.

Для того, чтобы в дереве, организованном по методу вложенных интервалов, кодирующихся цепными дробями, удалить элементы определенного поддерева, необходимо по рациональному коду корня рассматриваемого поддерева найти определяемый им интервал, в который попадают все рациональные коды данного поддерева. Для этого можно применить, используемый в алгоритме поиска элементов поддерева, способ определения интервала поддерева. После этого необходимо удалить все элементы дерева, находящиеся в данном интервале, и корень данного поддерева. По результатам использования, описанного выше алгоритма, SQL-запрос на удаление элементов определенного поддерева будет выглядеть следующим образом:

delete from nested_intervals where numerator/denominator > a/b and numerator/denominator <= (a+c)/(b+d);

delete from nested_intervals where numerator = a and denominator = b;

5. Метод объединяющий материализацию пути и список смежности

Объединенное отношение отличается от отношения метода материализации, тем, что поле, хранящее последний порядковый номер узла, заменено полем, которое будет хранить идентификатор родителя. Отказаться от поля SIBLING_MAX стало возможным из предположения использования, описанного в предыдущем подразделе метода к кодированию материализованного пути (с помощью уникальных идентификаторов).

Таблица

Реляционное отношение объединенной модели материализации пути и списка смежности

Поле	**Тип**	**Описание**
Id	int4	Идентификатор элемента
Path	varchar(500)	Материализованный путь
Parent_id	int4	Идентификатор родителя

Операция добавления элемента в дерево.

Для добавления элемента в дерево организованное по предлагаемому методу, необходимо знать идентификатор нового элемента, идентификатор родителя и материализованный путь родителя. Пусть *ID* есть идентификатор нового элемента, *PARENT_ID* есть идентификатор родителя, а *PARENT_PATH* есть материализованный путь его родителя, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на вставку нового элемента будет выглядеть следующим образом:

insert into tree (id, path) values (ID, PARENT_PATH || '.' || ID, PARENT_ID);

Следует заметить, что в данном случае материализованный путь является объединением двух строк: *PARENT_PATH || '.' || ID*, где *PARENT_PATH* – строка материализованного пути родителя, а *ID* – преобразованный в строку идентификатор нового элемента. Необходимо

отметить, что идентификатор элемента является числовым значением и его преобразование в строку может быть различным (см. выше описанный, Способ кодирования шага пути).

Операция удаления элемента из дерева.

Как было отмечено ранее, при удалении элемента из дерева необходимо не только удалить элемент, но и перенастроить связи оставшихся элементов, для сохранения древовидной иерархии. Для предлагаемой модели не был найден прямой способ удаления элемента. Однако выполнение рассматриваемой операции может быть выполнено следующим образом: необходимо переместить все поддеревья, корнями которых являются сыновья удаляемого элемента, так, чтобы родителем этих элементов стал родитель удаляемого элемента. Операции перемещения дерева и поиска сыновей рассмотрены ниже.

Операция поиска родителей элемента.

Процесс поиска родителей определенного элемента дерева при организации дерева по предлагаемому методу не требует выполнения запроса, так как по описанному методу кодирования идентификаторы всех родителей элемента уже представлены в материализованном пути. Пусть материализованный путь для элемента равен $C_1.C_2....C_{K-1}.C_K$, тогда идентификаторы его будут равны: $\{ C_1.C_2....C_{K-2}.C_{K-1}\}$.

Операция поиска сыновей элемента.

Поиск сыновей элемента дерева при его организации по предлагаемому методу будет идентичен поиску по методу списка смежности. Объединение моделей материализации пути и модели списка смежности было выполнено, именно, для увеличения эффективности выполнения операции поиска сыновей. Теперь, после объединения, поиск сыновей является достаточно простым и требует выполнения одного SQL-запроса. Пусть *ID* есть идентификатор элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск сыновей элемента будет выглядеть следующим образом:

select id from tree where parent_id=ID;

Операция поиска элементов поддерева.

Для решения задачи поиска элементов поддерева необходим один SQL-запрос. Пусть *PATH* есть материализованный путь до элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск элементов поддерева будет выглядеть следующим образом:

select id from tree where path like 'PATH.%';

Операция перемещения поддерева.

Любое изменение иерархии требует изменения всех элементов, кодирующих иерархическое отношение. В нашем случае, в объединенной модели иерархия кодируется как по методу материализации пути, так и по методу списка смежности. Поэтому при перемещении дерева необходимо как изменить питии элементов так и идентификаторы их родителей.

Перемещение поддерева внутри иерархии в случае, когда дерево организовано по методу материализации пути, требует выполнения достаточно сложного SQL-запроса, который затрагивает все элементы поддерева, так как перемещение поддерева требует изменения значений материализованных путей у всех элементов поддерева. Что касается идентификаторов родителей элемента, то при перемещении дерева необходимо изменить такой идентификатор только у корня перемещаемого дерева. Пусть материализованный путь корня перемещаемого поддерева равен $C_1.C_2....C_{K-1}.C_K$, а *ROOT_ID* его идентификатор. Соответственно в множестве префиксов каждого из материализованных путей всех его потомков будет существовать префикс $C_1.C_2....C_{K-1}.C_K$. Пусть материализованный путь элемента, который будет новым родителем корня перемещаемого дерева равен $A_1.A_2....A_{M-1}.A_M$, а *NEW_ID* его идентификатор. Тогда для перемещения выбранного поддерева необходимо заменить у всех элементов поддерева префикс $C_1.C_2....C_{K-1}.C_K$ на $A_1.A_2....A_{M-1}.A_M.A_{M+1}$, где A_{M+1} является новой частью пути, которая определяет положение корня переносимого поддерева среди сыновей нового корня. И у корня дерева

изменить значение его родителя на новое. Для перемещения поддерева может быть предложен следующий алгоритм:

1: Определяем новый префикс для элементов поддерева равный $A_1.A_2....A_{M-1}.A_M.C_K$.

2: Перемещаем элементы поддерева. Изменяем старый префикс элементов поддерева равный $C_1.C_2....C_{K-1}.C_K$ *на* $A_1.A_2....A_{M-1}.A_M.C_K$. *Данную операцию можно провести посредством выполнения следующего SQL-запроса:*

update tree set path = overlay(path placing '$A_1.A_2....A_{M-1}.A_M.C_K$*' from position('*$C_1.C_2....C_{K-1}.C_K$*' in path) for char_length('*$C_1.C_2....C_{K-1}.C_K$*')) where path='* $C_1.C_2....C_{K-1}.C_K$*' or path like '* $C_1.C_2....C_{K-1}.C_K$*.%';*

3: Изменяем значение идентификатора родителя у корня перемещаемого дерева. Данную операцию можно провести посредством выполнения следующего SQL-запроса:

update tree set parent_id = NEW_ID where id = ROOT_ID;

Операция удаления поддерева.

Удаление элементов поддерева с определенным элементом в качестве корня, как и поиск элементов поддерева является достаточно простой операцией на дереве, организованном по предлагаемому методу, и выполняется на основе модели материализации пути. Для решения задачи поиска элементов поддерева необходим один SQL-запрос. Пусть *PATH* есть материализованный путь до элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск элементов поддерева будет выглядеть следующим образом:

delete from tree where path='PATH' or path like 'PATH.%';

6. Метод транзитивного замыкания

В данном методе в реляционном отношении сохраняется транзитивное замыкание реляционного отношения списка смежности. Транзитивным замыканием отношения *R* является отношение R^+ [114]. Тем самым, сохраняя транзитивное замыкание иерархического отношения, мы сохраняем пары (*предок, потомок*). Для возможности выполнения всех иерархических операций в реляционное отношении транзитивного замыкания в каждый кортеж вводят поле, определяющее степень отношения *R* между элементами данного кортежа. Степень иерархического отношения является, по сути, расстоянием между элементами отношения или количеством дуг между ними в дереве.

Таблица

Реляционное отношение для транзитивного замыкания

Поле	**Тип**	**Описание**
descendant	int4	Идентификатор потомка
Ancestor	int4	Идентификатор предка
Distance	int4	Расстояние в иерархии между двумя элементами (количество дуг)

Операция добавления элемента в дерево.

Для добавления элемента в дерево, организованное как транзитивное замыкание списка смежности, необходимо составить список предков элемента элемента, сыном которого будет новый элемент. Затем для каждого из предков необходимо создать транзитивную связь. Пусть *ID* есть идентификатор нового элемента, *ANCESTOR_ID* есть идентификатор предка, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на вставку новой транзитивной связи будет следующим:

insert into tree values (ID, ANCESTOR_ID, distance);

Операция удаления элемента из дерева.

Как было сказано ранее, при удалении элемента из дерева необходимо не только удалить элемент, но и перенастроить связи оставшихся элементов, для сохранения древовидной иерархии. Так, при организации дерева как транзитивного замыкания списка смежности, необходимо в транзитивных связях, где участвуют элементы поддерева, в котором удаляемый элемент является корнем, сыновей удаляемого элемента нового родителя, уменьшить *distance* на единицу. После этого можно удалить все транзитивные связи, в которых участвует удаляемый элемент. Пусть *ID* есть идентификатор удаляемого элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на удаление элемента будет выглядеть следующим образом:

delete from tree where ancestor=ID or descendant=ID;

Операция поиска родителей элемента.

Для нахождения родителей определенного элемента дерева при организации дерева как транзитивного замыкания списка смежности необходимо выполнить один SQL-запрос. Пусть *ID* есть идентификатор текущего элемента, а *tree* является таблицей, в которой хранится дерево, тогда для поиска всех родителей элемента с идентификатором *ID* вызвать следующий SQL-запрос:

select id from tree where descendant = ID order by distance asc;

Операция поиска сыновей элемента.

Нахождение сыновей элемента дерева при его организации как транзитивного замыкания списка смежности является достаточно простым и требует выполнения одного SQL-запроса. Пусть *ID* есть идентификатор элемента, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск сыновей элемента будет выглядеть следующим образом:

select id from tree where ancestor=ID and distance=1;

Операция поиска элементов поддерева.

Нахождение элементов поддерева при дерева организации как транзитивного замыкания списка смежности является достаточно простым и

требует выполнения одного SQL-запроса. Пусть *ID* есть идентификатор корня поддерева, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на поиск элементов поддерева будет выглядеть следующим образом:

select id from tree where ancestor=ID;

Операция перемещения поддерева.

При перемещении поддерева в дереве, организованном в виде транзитивного замыкания списка смежности необходимо заменить часть транзитивного замыкания, которое кодирует перемещаемое поддерева. А именно, удаляется часть транзитивного замыкания, в элементах которой в качестве предка выступают элементы, которые находятся выше по иерархии по отношении корня перемещаемого поддерева. После этого, для элементов перемещаемого поддерева устанавливаются транзитивные связи, в которых в качестве предка выступают новые предки корня рассматриваемого поддерева.

Операция удаления поддерева.

Для того чтобы удалить поддерево из дерева, организованного в виде транзитивного замыкания списка смежности необходимо составить список элементов поддерева и затем удалить все транзитивные связи, в которых участвуют данные элементы. Пусть *ID* есть идентификатор элемента поддерева, а *tree* является таблицей, в которой хранится дерево, тогда SQL-запрос на удаление транзитивных связей будет следующим:

delete from tree where ancestor=ID or descendant=ID;

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Agile Data Home Page. [Электронный ресурс] – Режим доступа: http://www.agiledata.org

2. Amber Scott W., The Process of Database Refactoring, [Электронный ресурс] – Режим доступа: http://www.agiledata.org/databaseRefactoring.html

3. Ambler Scott, Sadalage Pramod, Database refactoring. [Электронный ресурс] – Режим доступа: http://www.databaserefactoring.com

4. Argawal Rakesh, Borgida Alexander, Jagadish H.V., Efficient management of transitive relationships in large data and knowledge bases // Proceedings of the 1989 ACM SIGMOD international conference on Management of data, Portland, Oregon, United States, 1989, pp. 253-262.

5. Ben-Gan Itzik, Maintaining Hierarchies. [Электронный ресурс] – Режим доступа: http://www.winnetmag.com/SQLServer/Articles/ArticleID/8826/

6. Beng Chin Ooi, Jiawei Han, Hongjuin Lu, Kian Lee Tan. Index nesting – an efficient approach to indexing in object-oriented databases // The VLDB Journal, 1996.

7. Bjorn Regnell, Requirements Engineering with Use Cases – a Basis for Software Development, Ph.D. Thesis, Department of Communication Systems, Lund University, 1999.

8. Brandon Daniel, Recursive database structures // Journal of Computing Sciences in Colleges, Volume 21 , Issue 2, 2005, pp. 295 – 304.

9. Brian Marick, Methodology Work Is Ontology Work, marick@visibleworkings.com

10. Business directory search engine. – US Patent number: 6523021; Inventors: Monberg; James C. (Seattle, WA); Mariani; Rico (Kirkland, WA); Staab; Sanford A. (Woodinville, WA); Applicant: Microsoft Corporation (Redmond, WA); Publication date: 2003-02-18.

11. BUSINESS DIRECTORY. – Inventors: INSTANT PAGES PTY LTD (AU), SUDHINDRA RAO (AU); Patent number: WO0201406; Publication date: 2002-01-03. [Электронный ресурс] – Режим доступа: http://v3.espacenet.com/textdes?DB=EPODOC&IDX=WO0201406&F=0&QPN=WO0201406

12. Celko Joe, Hierarchical SQL. – 2004. [Электронный ресурс] – Режим доступа: http://www.onlamp.com/pub/a/onlamp/2004/08/05/hierarchical_sql.html

13. Celko Joe, Trees and Hierarchies in SQL for Smarties .- Morgan Kaufmann 2004 .- 240pp.-

http://www.dbazine.com/ofinterest/oi-articles/celko24

http://www.sqlsummit.com/AdjacencyList.htm

14. Celko Joe, Trees in SQL // Intelegent Enterprise. [Электронный ресурс] – Режим доступа: http://www.intelligententerprise.com/001020/celko.jhtml?_requestid=697912

15. Celko Joe, Some answers to some common questions about SQL trees and hierarchies. http://www.intelligententerprise.com/001020/celko1_2.jhtml

16. Codd E.F. A Relational Model of Data for Large Shared Data Banks. Comm. Of the ACM< 1970, v.13, no. 6, pp. 377-387. (Пер.: Кодд. Е.Ф. Реляционная модель для больших совместно используемых банков данных // СУБД. – 1995. - №1. –С. 145-160.)

17. Damiani E., Fugini M.G., Bellettini C. A hierarchy-aware approach to faceted classification of object-oriented components // ACM Transactions on Software Engineering on Methodology, Vol. 8, No. 3, July 1999.77

18. Dan Alexandru Pescaru, Bridging the Gap between Object Oriented Modeling and Implementation Languages Using a Meta-Language Approach, PhD Thesis, Faculty of Automation and Computer Science ”POLITEHNICA” UNIVERSITY OF TIMISOARA, - Timisoara. 23 October 2003.

19. Deleurme Shawn, A Nested Set Implementation in Java and PostgreSQL [Электронный ресурс] – Режим доступа: http://threebit.net/tutorials/nestedset/tutorial1.html

20. Donald Bradley Roberts, Practical analysis for refactoring, Ph.D. Thesis, the Graduate college of University of Illinois at Urbana-Champaign, 1999.

21. Dong G., Libkin L., Su J. and Wong L. Maintaining the transitive closure of graphs in SQL. In Int. J. Information Technology, 1999. [Электронный ресурс] – Режим доступа: http://citeseer.ifi.unizh.ch/dong99maintaining.html

22. Dong Gouzhu, Su Jianwen. Incremental Maintenance of Recursive Views Using Relational Calculus/SQL // SIGMOD Record, Vol. 29, No. 1, March 2000.

23. Eric W. Weisstein. "Linear Fractional Transformation." From MathWorld--A Wolfram Web Resource. [Электронный ресурс] – Режим доступа: http://mathworld.wolfram.com/LinearFractionalTransformation.html

24. Forbes Dennis, Versatile High Performance Hierarchies in SQL Server: How to implement the Nested Set Model [Электронный ресурс] – Режим доступа: http://www.yafla.com/papers/sqlhierarchies/sqlhierarchies.htm

25. Fowler Martin, Pramond Sadalage, Evolutionary Database Design, 2003. [Электронный ресурс] – Режим доступа: http://www.martinfowler.com

26. Gassiep Naz, Working with hierarchies in relational databases [Электронный ресурс] – Режим доступа: http://old.mrnaz.com/notes/nestedsets/

27. Goodman Nathan, Oded Shmueli. Tree queries: a simple class of relational queries // ACM Transactions of database systems, Vol. 7, No. 4, December 1982.

28. Haughey Tom, Modeling Hierachies [Электронный ресурс] – Режим доступа: http://www.tdan.com/special031.htm

29. Hillyer Mike, Managing Hierarchical Data in MySQL [Электронный ресурс] – Режим доступа: http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

30. Jagadish H.V. Incorporating hierarchy in a relational model of data // SIGMOD Report, 1989.

31. Jonsson Lennart, Representing Trees in a relational DB [Электронный ресурс] – Режим доступа: http://fungus.teststation.com/~jon/treehandling/TreeHandling.htm

32. Larry McCay, If(extremeProgramming.equals(scientificMethod)) Programming as theory building, [Электронный ресурс] – Режим доступа http://www.java.sys-con.com

33. Libkin Leonid, Wong Limsoon. SQL Can Maintain Polynomial-Hierarchy Queries, 1997. [Электронный ресурс] – Режим доступа: http://citeseer.ifi.unizh.ch/libkin97sql.html

34. Lu Hongjun. New strategies for computing the transitive closure of a database relation // Proceedings of the 13th VLDB Conference, Brighton, 1987.

35. Ltree PostgreSQL module for materialized path trees [Электронный ресурс] – Режим доступа: http://www.sai.msu.su/~megera/postgres/gist/ltree/

36. Mackey Aaron, Relational Modeling of Biological Data: Trees and Graphs, 2002. [Электронный ресурс] – Режим доступа: http://www.oreillynet.com/pub/a/network/2002/11/27/bioconf.html

37. Marcio Lopes Cornelio, Refactorings as formal refinements, Ph.D. Thesis, Universiade Federal de Pernambuco, Centro de Informatica, Recife, 2004.

38. Mathews, J. The Moebius Transformation. [Электронный ресурс] – Режим доступа: http://www.ecs.fullerton.edu/~mathews/fofz/mobius/

39. More Trees & Hierarchies in SQL. [Электронный ресурс] – Режим доступа: http://www.sqlteam.com/item.asp?ItemID=8866

40. NETWORK DIRECTORY FOR BUSINESS PROCESS INTEGRATION OF TRADING PARTNERS, - Patent number: WO2004036348; Publication date: 2004-12-29; Inventors: CLARK GREGORY SCOTT; SRINIVASAN ANDRE; BURGESS DAVID; SCHNIEDER DAVID; PILZ GILBERT; SRINIVASAN KARTHIK; Applicant: E2OPEN LLC (US). [Электронный ресурс] – Режим доступа: http://v3.espacenet.com/textdes?DB=EPODOC&IDX=WO2004036348&F=0&QPN=WO2004036348

41. Online business directory with predefined search template for facilitating the matching of buyers to qualified sellers. – US Patent number: 6189003; Inventors: Leal; Fernando (Chicago, IL); Applicant: WynWyn.com Inc. (Chicago, IL); Publication date: 2001-02-13

42. Refactoring Home Page. [Электронный ресурс] – Режим доступа: http://www.refactoring.com

43. Rick Mugridge, Test Driven Development and the Scientific Method, Department of Computer Science, University of Auckland, New Zealand, r.mugridge@auckland.ac.nz

44. Sander Tichelaar, Modeling Object-Oriented Software for Reverse Engineering and Refactoring, Inauguraldissertation der Philosophisch-naturwissenschaftlichen Fakultдt, der Universitдt Bern, Bern, 14. Dezember 2001

45. SEARCH QUERY CATEGORIZATION FOR BUSINESS LISTINGS SEARCH. – Patent number: WO2004114162; Publication date: 2004-12-29; Inventors: MALPANI RADHIKA (US); MITTAL VIBHU (US); Applicant: GOOGLE INC (US); MALPANI RADHIKA (US); MITTAL VIBHU (US). [Электронный ресурс] – Режим доступа: http://v3.espacenet.com/textdes?DB=EPODOC&IDX=WO2004114162&F=0&QPN=WO2004114162

46. Storing Hierarchical Data in a Database [Электронный ресурс] – Режим доступа: http://www.sitepoint.com/print/1105

47. The LAN-simulation: A Refactoring Teaching Example Serge Demeyer, Filip Van Rysselberghe, Tudor Gırba, Jacek Ratzinger, Radu Marinescu, Tom Mens, Bart Du Bois, Dirk Janssens, Stephane Ducasse, Michele Lanza,Matthias Rieger, Harald Gall, Mohammad El-Ramly

48. Tree In Sql. [Электронный ресурс] – Режим доступа: http://c2.com/cgi/wiki?TreeInSql. – Загл. с экрана.

49. Troels' links: Relational database systems. [Электронный ресурс] – Режим доступа: http://troels.arvin.dk/db/rdbms/links. - Загл. с экрана.

50. Tropashko Vadim, Integer Labeling in Nested Intervals Model [Электронный ресурс] – Режим доступа: http://www.dbazine.com/oracle/or-articles/tropashko6

51. Tropashko Vadim, Nested Intervals with Farey Fractions .- 2004 [Электронный ресурс] – Режим доступа: http://arxiv.org/html/cs.DB/0401014

52. Tropashko Vadim, Nested Intervals Tree Encoding with Continued Fractions // SIGMOD Record June 2005, Volume 34, Number 2 [Электронный ресурс] – Режим доступа: http://arxiv.org/abs/cs.DB/0402051

53. Tropashko Vadim, Relocating Subtrees in Nested Intervals Model .- 2003 [Электронный ресурс] – Режим доступа: http://www.dbazine.com/tropashko5.shtml

54. Tropashko Vadim, Trees in SQL: Nested Sets and Materialized Path .- 2003 [Электронный ресурс] – Режим доступа: http://www.dbazine.com/tropashko4.shtml

55. William F. Opdyke, Refactoring object-orientired frameworks, Ph.D. Thesis, the Graduate college of University of Illinois at Urbana-Champaign, 1992.

56. Wood William A., Kleb William L.. Exploring XP for Scientific Research, IEEE Software, vol. 20, no. 3, pp. 30-36, May/June, 2003.

57. XML для профессионалов, Мартин Д., Бирбек М., Кэй М., Лозген Б., Пинок Д., Ливингстон С., Старк П., Уильямс К., Андерсон Р., Мор С., Балилес Д., Пит Б., Озу Н. – М: Издательство «Лори», 2001.

58. Алистер Коберн, Современные методы описания функциональных требований к системам, Издательство «Лори», 2002

59. Андерсон, Джеймс А., Дискретная математика и комбинаторика: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 960 с. : ил. – Парал. тит. англ.

60. Архангельский А. Древовидные (иерархические) структуры данных в реляционных базах данных, 2005. [Электронный ресурс] – Режим доступа: http://www.az-design.ru/Support/DataBase/DBTreeToc.shtml

61. Ахо А.В., Хопкрофт, Д., Ульман Д.Д. Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 384 с.

62. Бен-Ган Ицик, Иерархические структуры, не требующие сопровождения//SQL Server Magazine №10, 2001. [Электронный ресурс] – Режим доступа: http://www.osp.ru/win2000/sql/967_print.htm

63. Берзин Вячеслав, Технология нагрузочного тестирования информационных систем с большим объемом данных // ORACLE Magazine, декабрь 2004. [Электронный ресурс] – Режим доступа: http://www.oracle.com/global/ru/oramag/dec2004/gen_load_test_it.html

64. Бизнес//Википедиа – свободная энциклопедия. [Электронный ресурс] – Режим доступа: http://ru.wikipedia.org/wiki/%25D0%2591%25D0%25B8%25D0%25B7%25D0%25BD%25D0%25B5%25D1%2581

65. Бизнес-справочник «Товары и услуги Хакасии и юга Красноярского края» [Электронный ресурс] – Режим доступа: http://www.sib-info.ru

66. Буч. Г., Рамбо Д., Джекобсон А. Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. 496 с.

67. Буч. Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. М.: ДМК, 2000. 432 с.

68. Буч.Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд.СПб.: Невский диалект, 1998. 560 с.

69. Виноградов С. А., Моделирование иерархических объектов (04.06.2001 г.) [Электронный ресурс] – Режим доступа: http://www.citforum.ru/database/articles/tree.shtml

70. Гаврилов В.Р., Иванова Е.Е., Морозова В.Д. Кратные и криволинейные интегралы. Элементы теории поля : Учеб. для вузов / Под ред. В.С. Зарубина, А.П. Крищенко. – М.: Изд-во МГТУ им. Н.Э.Баумана, 2001. – 492 с.

71. Гайнов А.Т. Теория чисел. Часть 1. Методическое пособие. Новосибирский Государственный университет. Механико-Математический факультет. 1999.

72. Гайнов А.Т. Теория чисел. Часть 2. Методическое пособие. Новосибирский Государственный университет. Механико-Математический факультет. 1999.

73. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. – СПб: Питер, 2001. – 368 с.: ил.

74. Гладков М.В. Методы и средства обработки информации в реляционных базах данных : Автореферат диссертации на соискание ученой

степени канд. техн. наук :05.13.01,05.13.11/ М.В. Гладков. -Пенза, 2004.-21 с. :ил.

75. Гмурман, В.Е. Теория вероятностей и математическая статистика. – М.: Высш. Шк., 2005. – 479 с.: ил.

76. Голованов Михаил, Иерархические структуры данных в реляционных БД// RSDN Magazine #0, 2002. [Электронный ресурс] – Режим доступа: http://www.rsdn.ru/mag/main.htm

77. Гребенщиков Н.Н., Применение современных Интернет-технологий при разработке сайта ХГУ. Вестник Хакасского государственного университета им. Н.Ф. Катанова. Выпуск 4. Серия 1: Информатика – Абакан: Издательство Хакасского государственного университета им. Н.Ф Катанова, 2001. – 251 с.

78. Гребенщиков Н.Н., Пример подхода к процессу разработки интерактивного Интернет-сайта. Вестник Хакасского государственного университета им. Н.Ф. Катанова. Выпуск 4. Серия 1: Информатика – Абакан: Издательство Хакасского государственного университета им. Н.Ф Катанова, 2003. – 280 с.

79. Гребенщиков Н.Н., Гузов М., Развитие методов реализации древовидной иерархии в реляционных базах данных // Материалы конференции «Развитие средств и комплексов связи» – Новочеркасск, 2005.

80. Гребенщиков Н.Н., Гузов М., Критерии оценки методов реализации абстрактных типов данных в реляционной модели // Материалы конференции «Военная электроника: опыт использования и проблемы подготовки специалистов» – Воронеж, 2005.

81. Гребенщиков Н.Н., Способ сравнения эффективности методов реляционного моделирования иерархических структур.. Вестник Хакасского государственного университета им. Н.Ф. Катанова. Информатика – Абакан: Издательство Хакасского государственного университета им. Н.Ф Катанова, 2006.

82. Гребенщиков Н.Н., Сравнение эффективности методов реляционного моделирования древовидного классификатора бизнес-справочника. Вестник Хакасского государственного университета им. Н.Ф. Катанова. Информатика – Абакан: Издательство Хакасского государственного университета им. Н.Ф Катанова, 2006.

83. Гребенщиков Н.Н., Масштабируемое реляционно-иерархическое хранилище объектной модели. // Тезисы докладов конференции конкурса «Технологии Microsoft в теории и практике программирования» – НГУ, Новосибирск, 2006.

84. Гребенщиков Н.Н., Метод реляционного моделирования иерархий. // Информатика и проблемы телекоммуникаций – СибГУТИ, Новосибирск, 2006.

85. Гребенщиков Н.Н., Представление древовидной зависимости в реляционной базе данных // Программные продукты и системы. №1 (81), 2008 – НИИ ЦПС, Тверь, 2008

86. Громов Ю.Ю., Татаренко С.И. Введение в методы численного анализа: Курс лекций.- Тамбов. госуд. техн. ун-т. 2001.

87. Дейт К. Дж. Введение в системы баз данных, 6-е издание: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.

88. Долинко В.В., Основы экономики предприятия. [Электронный ресурс] – Режим доступа: http://www.ukrlib.boom.ru/ekpr/ekpredpr.html

89. Дьяконов В. Mathcad 8-12 для всех. Полное руководство. – Солон, 2005.

90. Закон Республики Хакасия «О государственной поддержке малого предпринимательства в Республике Хакасия» от 22.05.1996, №62.

91. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. – СПб.: БХВ-Петербург, 2003. – 1104 с.: ил.

92. Келин Ю.Е., Классификация предприятий. [Электронный ресурс] – Режим доступа: http://www.eabc.edu.ee/~jurikelin/

93. Кельтон Дэвид В., Лоу Аверилл М. Имитационное моделирование. Классика CS. 3-е изд. – СПб.: Питер; Киев: Издательская группа BHV, 2004. – 847 с.: ил.

94. Кирилов В.В. Основы проектирования реляционных баз данных. Учебное пособие. Санкт-Петербургский Государственный институт точной механики и оптики (технический университет). [Электронный ресурс] – Режим доступа: http://www.citforum.ru/database/dbguide/index.shtml

95. Классификатор//Новый словарь русского языка. [Электронный ресурс] – Режим доступа: http://www.rubricon.com

96. Классификация (систематизация)//Большая советская энциклопедия. [Электронный ресурс] – Режим доступа: http://www.rubricon.com

97. Книга А.С. Статистика: Учебное пособие/ Алт. госуд. технич. ун-т. И.И. Ползунова. – Барнаул: 2003. – 150 с. Издание 2-е, исправленное и переработанное.

98. Кнут, Дональд, Эрвин, Искусство программирования, том 1. Основные алгоритмы, 3-е изд. : Пер. с англ. Уч. Пос. – М. : Издательский дом «Вильямс», 2000. – 720 с.: ил.

99. Кнут, Дональд, Эрвин, Искусство программирования, том 2. Получисленные алгоритмы, 3-е изд. : Пер. с англ. Уч. Пос. – М. : Издательский дом «Вильямс», 2000. – 720 с.: ил.

100. Кнут, Дональд, Эрвин, Искусство программирования, том 3. Сортировка и поиск, 3-е изд. : Пер. с англ. Уч. Пос. – М. : Издательский дом «Вильямс», 2000. – 720 с.: ил. Кнут 3 том

101. Коберн А. Быстрая разработка программного обеспечения. М.: Лори, 2002. 314 с.

102. Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002. – 800с.: ил.

103. Коннолии Томас, Бегг Каролин, Страчан Анна, Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд. : Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 1120 с.

104. Кормен Т., Лейзерсон Ч., Ривест Р., Алгоритмы: построение и анализ / Пер. с англ. под ред. А Шеня. – М.: МЦНМО, 2002. -960 с.: 263 ил.

105. Куликов Алексей, Посадим дерево? [Электронный ресурс] – Режим доступа: http://dull.ru/2004/11/11/posadim_derevo/

106. Курчидис В. А., Чиркунов В. А., Назанский А. С. Эффективное представление древовидных структур в реляционных базах данных. / Ярославский государственный университет им. П. Г. Демидова. - Ярославль, 2003. - 12 с., библ. 8. - Рус. - Деп. в ВИНИТИ 27.02.2003, № 380-В2003.

107. Кэй М. XSLT. Справочник программиста. – Пер. с англ. –СПб: Символ-Плюс, 2002. – 1016 с., ил.

108. Макаров Е. Инженерные расчеты в MathCad. – СПб: Питер, 2003. – 448 с.

109. Макгрегор Дж., Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. – К.: ООО «ТИД «ДС», 2002.

110. Малый энциклопедический словарь Брокгауза и Ефрона. [Электронный ресурс] – Режим доступа:http://slovari.yandex.ru

111. Мартыненко С. Объемное тестирование на стадии выбора архитектуры, 2005. [Электронный ресурс] – Режим доступа: http://software-testing.ru/lib/martynenko/volume-tesing.htm

112. Мартыненко С. Терминология. Нагрузочное тестирование, 2005. [Электронный ресурс] – Режим доступа: http://software-testing.ru/lib/martynenko/term-perf-tesing.htm

113. Мишулович А. Проблема нагрузочного тестирования компонентов биллинговых систем, 2005. [Электронный ресурс] – Режим доступа: http://software-testing.ru/lib/mishulovin/billing-components-performance-testing.htm

114. Новиков Ф.А. Дискретная математика для программистов – СПб.: Питер, 2001.

115. Орлов С. Технологии разработки программного обеспечения. Учебное пособие. СПб.: Питер, 2003. – 480 с.

116. Постановление Правительства Российской Федерации от 10.11.2003 № 677 "Об общероссийских классификаторах технико-экономической и социальной информации в социально-экономической области".

117. Постановление Совета Министров Республики Хакасия «О положении о комитете по поддержке и развитию малого предпринимательства при Совете Министров Республики Хакасия и о фонде поддержки малого предпринимательства Республики Хакасия» от 10.01.1996, №8.

118. Постановление правительства Республики Хакасия «О реестре субъектов малого предпринимательства Республики Хакасия» от 17.07.1998, №114.

119. Сажин А.С. Сравнительный анализ различных подходов к обработке древовидных структур в SQL//Научная сессия МИФИ-2004, Секция: Технологии программных систем. [Электронный ресурс] – Режим доступа: http://www.library.mephi.ru/data/scientific-sessions/2004/2/058.pdf

120. Смирнов С.Н. Задворьев И.С. Работаем с Oracle. – Гелиос АРВ, 2002.

121. Смит Артур Б. Реляционные, древовидные и объектно-ориентированные базы данных//M Computing May/June 1996, v.4 n.2, стр.8. [Электронный ресурс] – Режим доступа: http://inftech.webservis.ru/it/database/oo/index.html

122. Справочник//Большая советская энциклопедия. [Электронный ресурс] – Режим доступа: http://www.rubricon.com

123. Таха, Хэмди, А. Введение в исследование операций, 6-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 912 с.: ил.

124. Фаулер Мартин, Рефакторинг: улучшение существующего кода. – СПб: Символ-Плюс, 2003. 432 с., ил.

125. Фаулер М. Архитектура корпоративных программных приложений. – М.: Издательский дом «Вильямс», 2004.

126. Фаулер Мартин. UML. Основы. 3-е издание. – Символ-Плюс, 2005.

127. Федеральным закон № 184-ФЗ "О техническом регулировании" от 27 декабря 2002 года.

128. Эд Барнфилд, Брайен Уолтерс, Программирование «клиент-сервер» в локальных вычислительных сетях. – М.: Информационно-издательский дом «Филинъ», 1997. – 424 с.

129. Экономические и финансовые словари от Глоссарий.ру. [Электронный ресурс] – Режим доступа: http://slovari.yandex.ru

130. Якобсон А., Буч. Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002. 496 с.