МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
БУРЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Институт математики и информатики
Кафедра прикладной математики

Курсовая работа

Моделирование деревьев в СУБД MySQL

Выполнил:	студент 4 курса группы 05230
	Саяпин Никита Александрович
Научный руководитель:	ст.преп. кафедры ПМ, к.ф-м.н
	Трунин Дмитрий Олегович
Научный консультант:	ст.преп. кафедры ИТ
	Хабитуев Баир Викторович

Улан-Удэ
2016

Оглавление

ВВЕДЕНИЕ

Дерево - это структура данных, которая представляет собой древовидную структуру в виде набора связанных узлов, и являющаяся связным графом, не содержащим в себе циклов[?].

Данная структура благодаря своей простоте и удобству получила широкое применение в практических целях. В настоящее время многие информационные системы применяют в работе деревья. Они предназначены для хранения информации о списках друзей, комментариев к какой-либо новостной статье, для реализации списка участников бонусных программ и других задач. Однако при создании структур данного класса программисты чаще всего сталкиваются с проблемой быстродействия кода. Не секрет, что при достаточно большом объеме данных многие базы данных, хранящие в себе деревья, начинают проявлять сбой в работе. Поэтому основной задачей разработчиков является разработка таких программных систем для работы с данными, которые позволяют выполнить операции для работы с деревьями за как можно меньшее время.

Целью работы является реализация системы для генерации структур типа «Деревья» на языке программирования PHP[8, 4] с использованием системы управления базами данных MariaDB, которая является свободно распространяемой версией широко применяемой на практике системы управления базами данных MySQL[7, 10].

В рамках курсовой работы были поставлены следующие задачи:

1. Изучение и рассмотрение основных структур хранения деревьев в базах данных.

2. Программная реализация основных операций для работы с данными, хранящимися в различных типах дереьев.

3. Реализация и тестирование программного кода.

Курсовая работа состоит из введения, трех глав, заключения, списка литературы и приложения.

Глава 1

Основные структуры хранения деревьев в базе данных

1.1. Введение в хранение простых деревьев

Рассмотрим задачу хранения данных в дереве на конкретном примере.

Допустим, что мы разрабатываем некоторую систему хранения комментариев, в которой каждый пользователь может оставлять комментарии или отвечать на уже написанный комментарий. Перед нами стоит задача отслеживания цепочки ответов пользователей в данной системе, в котором каждый комментарий в зависимости от ситуации может ссылаться на один из комментариев или быть независмым от других комменатриев. Для хранения комменатриев мы используем базы данных, а для реализации связей применяются структуры деревьев. Однако тут же возникает следующий вопрос: каким образом можно хранить связи между элементами в базе данных?

Существуют четыре основные структуры хранения деревьев в базах данных[5]:

1. Список смежности(Adjacency List), хранящий информацию об узле и его родителе;

2. Перечисление путей(Materialized Path), который в каждом узле содержит информацию о пути от вершины дерева к данному узлу;

3. Вложенные множества(Nested Sets), структура, каждый элемент которой имеет два номера, которые идентифицируют ее как узел дерева, а также уровень вложенности соответствующего узла;

4. Таблица замыканий(Closure Table), структура, которая реализуется с помощью двух таблиц в базе данных, и содержит в каждом узле информацию не только о родителях данного узла и уровне вложенности, но и информацию о все родительской ветке для данного узла.

В рамках курсовой работы были реализованы все четыре основных метода реализации дереьев в среде MySQL/MariaDB. Помимо этого, описаны

преимущества и недостатки каждой из структур.

1.2. Информация об основных структурах реализации деревьев

В данной подглаве мы рассмотрим основные структуры хранения деревьев в базах данных, а также рассмотрим проектирование таблиц базы данных для СУБД MySQL.

В таблицах СУБД MySQL приняты соответствующие обозначения полей:

1.2.1. Список смежности

Эта структура содержит в себе представление о коллекции списков вершин, иначе говоря, каждой вершине дерева соответствует список, состоящий из узлов, которые связаны с данным[2]. Для того, чтобы реализовать ее, нам достаточно знать лишь информацию о том, к какому узлу ссылается текущий узел, или другими словами, связь между узлом и его родителем.

Соответственно, для практической реализации данной структуры нам достаточно занести в соответствующую таблицу базы данных два поля: идентификатор узла дерева и идентификатор его родителя[3].

Метод является одним из наболее простых для реализации деревьев. На практике он применяется во многих задачах, в частности, для хранения списка непосредственного списка друзей, списка комментариев и многих других задач, которые применяют для реализации древовидные структуры. Для удобства работы с методом доработаем таблицу, добавив в него два соответствующих поля: поле для хранения имени пользователя, а также поле статуса пользователя, которое показывает, заблокирован ли пользователь в системе или нет. Данное решение более предпочтительно, чем удаление пользователя из дерева, так как позволяет в кратчайшие сроки при возможности восстановить его. В таблице 1.1 показана изображена структура таблицы базы данных, реализующая данную структуру.

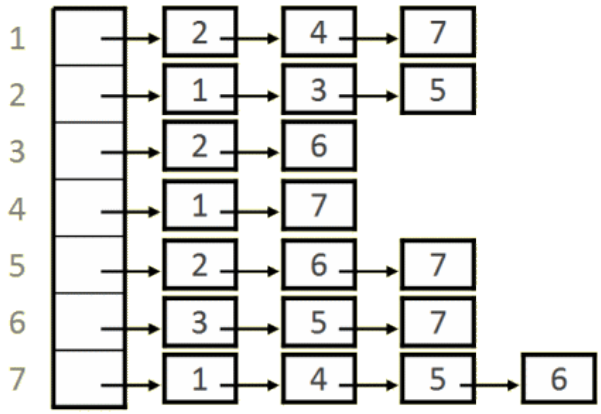

Рис. 1.1: Примерный графический вид структуры "Список смежности"в таблице СУБД MySQL

Таблица 1.1: Структура таблицы базы данных, реализующей структуру 'Список смежности'

Атрибут	Тип	Описание
user_id	int(11)	Идентификатор узла в структуре решения
parent_id	int(11)	Идентификатор родительского узла
user_name	varchar(255)	Имя узла
user_status	char(1)	Статус(свойство) узла

1.2.2. Перечисление путей

Данный метод экономно решает задачу извлечения предков заданного узла в дереве. Он хранит в себе поле, которое в каждом узле содержит путь, который начинается с родительского узла, не имеющего предков, и заканчивается в данном узле. Между ними находятся узлы, которые соединяют родительский узел с текущим. Данный метод удобен для организации иерархических структур, с его помощью можно легко вывести все комментарии к записи или каталог всех товаров, которые продаются на предприятии. Реализация алгоритма перечисления путей аналогична алгоритму списка смежности, только вместо поля хранения информации о родителе узла мы добавляем поле, в котором содержится путь для данного узла.

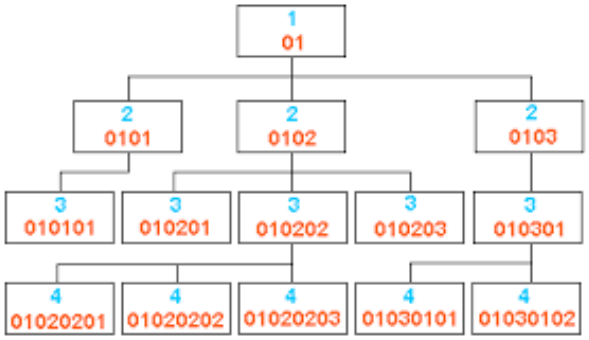

Рис. 1.2: Примерный графический вид структуры "Список смежности"в таблице СУБД MySQL

Таблица 1.2: Вид метода хранения деревьев 'Перечисление путей' в таблице СУБД MySQL

Атрибут	Тип	Описание
user_id	int(11)	Идентификатор узла в структуре решения
user_name	varchar(255)	Имя узла
user_path	varchar(255)	Путь к узлу, начиная с корня дерева
user_status	char(1)	Статус(свойство) узла

В процессе работы с данной структуры можно столкнуться с некорректной сортировкой путей. Дело в том, что поле, которое хранит путь в таблице базы данных, является строковым. Поэтому оно сортирует поле в алфавитном порядке, а не в числовом. Так например, при сортировке данных по пути после узла с полем '/1' может стоять узел, содержащий в себе путь '/1/10', а не '/1/2'. Для того, чтобы избежать подобной проблемы, применяются различные доработки базы данных. Одним из наиболее простых способов решения является добавление параметра UNSIGNED ZEROFILL для идентификатора данного узла. Данный параметр автоматически дополняет нулями данный идентификатор до того количества символов, которым ограничено данное поле в таблице. Тогда путь будет содержать информацию о связях узлов с учетом данного параметра. Недостатком данной доработки являются ограничения на длину идентификатора пользователя и пути, которые не позволяют вставлять узлы ниже определенного уровня, а также узлы, номер которых содержит в себе цифр больше, чем может

вместить в себя идентификатор.

1.2.3. Вложенные множества

Данное решение по хранению деревьев позволяет хранить информацию с каждым узлом, принадлежащим множеству его потомков, а не его непосредственным родителем. Данную информацю можно получить с помощью кодирования каждого узла с помощью двух номеров, левого и правого. Эти номера задаются следующим образом: левый номер меньше, а правый соответственно больше всех номеров дочерних объектов. Способом назначений левого и правого номера в этом методе состоит в выполнении прохода по всему дереву. Каждому узлу присваиваются значения левого номера с приращением при спуске по ветви дерева и значения правого номера при обратном подъеме по ветви[11]. При желании в таблицу можно также добавить и

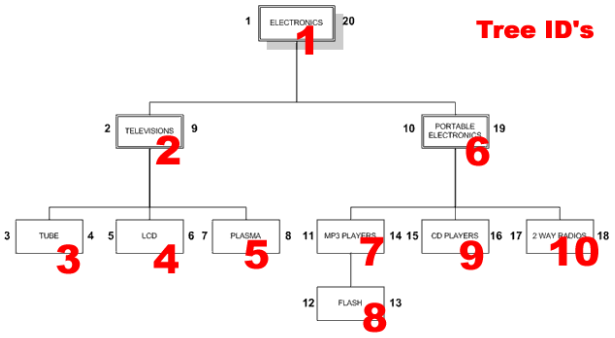

Рис. 1.3: Примерный графический вид структуры "Вложенные множества"в таблице СУБД MySQL

уровень вложенности текущего узла. Это поле показывает положение узла в текущей структуре дерева и определяется по минимальному количеству переходов от корня дерева до данного узла. Например, корневой узел ’1’ имеет уровень вложенности ’1’, а узел с идентификатором ’3’ , родителем которого является корневой узел ’1’ , имеет уровень вложенности "2".

Рисунок 1.3 показывает структуру данного решения в базе данных MySQL.

Таблица 1.3: Вид метода хранения деревьев 'Вложенные множества' в таблице СУБД MySQL

Атрибут	Тип	Описание
user_id	int(11)	Идентификатор узла в структуре решения
user_name	varchar(255)	Имя узла
left_number	int(11)	Левый номер узла
right_number	int(11)	Правый номер узла
level	char(1)	Уровень вложенности узла
user_status	char(1)	Статус(свойство) узла

1.2.4. Таблица замыканий

Этот метод является простым способом хранения иерархий в базе данных [9]. В отличие от остальных методов, этот метод реализуется с помощью двух таблиц, а не одной. В первой таблице хранится информация об узлх дерева(в частности, там может зранится информация об имени и статусе узла). Во второй же таблице содержится вся информация о связях между элементами дерева, причем информация о связи между родителями и потомками заносится даже в том случае, если их разделяют несколько уровней, то есть таблица содержит информацию не только о непосредственном родителе данного узла, но и о предках данного родителя до корневого узла.

Помимо всего прочего, в таблице базы данных хранится ссылка на данный элемент, которая позволяет ссылаться на текущий узел. Так, например, нам нужно занести в базу данных информацию о связах узла 17, чьим предком является узел 10, который является ребенком узла 5, являющегося сыном коренного узла. Тогда в таблицу занесутся следующие записи(таблица 1.4):

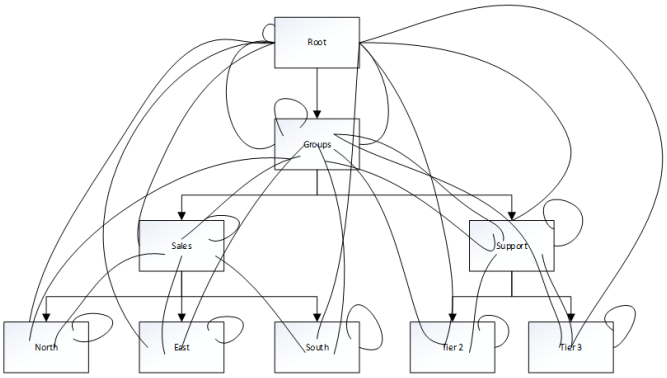

Рис. 1.4: Вид метода хранения деревьев "Таблица замыканий"в таблице СУБД MySQL

Таблица 1.4: Вид вставляемых элементов в таблицу связей для метода 'Таблица замыканий'

user id	parent id	level
17	1	4
17	5	3
17	10	2
17	17	1

В таблицах 1.5 и 1.6 изображено представление двух таблиц, реализующих данный метод хранения деревьев- таблицы, содержащей информацию об узлах дерева и таблицы,в которой лежит информация о связях между элементами соответственно.

Таблица 1.5: Таблица узлов метода хранения деревьев 'Таблица замыканий' в таблице СУБД MySQL

Атрибут	Тип	Описание
user_id	int(11)	Идентификатор узла в структуре решения
user_name	varchar(255)	Имя узла
user_status	char(1)	Статус(свойство) узла

Таблица 1.6: Таблица связей метода хранения деревьев 'Таблица замыканий' в таблице СУБД MySQL

Атрибут	Тип	Описание
num_id	int(11)	Идентификатор записи в базе данных
user_id	varchar(255)	Идентификатор узла
parent_id	int(11)	Идентфикатор родительского узла
level	char(1)	Уровень связи между узлами

Глава 2

Основные операции для деревьев

В рамках курсовой работы была реализована система для работы со структурами типа "Деревья"с использованием интерпретатора скриптов PHP и системы управления базами данных MariaDB 10.1.16. Программный код для каждой из четырех структур представлен в приложении.

В рамках курсовой работы были рассмотрены и реализованы следующие операции по работе со структурами типа "Деревья":

1. Создание нового дерева взамен существующего(функция create).
2. Вставка узла в дерево(функция add).
3. Удаление узла(функция delete).
4. Блокирование и разблокирование узла(функции block и unlock).
5. Нахождение подчиненной ветки для для данного узла(функция child_branch).
6. Нахождение непосредственных детей для данного узла(функция child).
7. Нахождение родительской ветки для данного узла(функция parent_branch).
8. Нахождение полного дерева для текущей структуры(функция tree).

Данная глава состоит из двух подглав. В первой главе рассмотрены базовые операции для данного узла(создание дерева, вставка, удаление, блокирование и перемещение узла). Во второй подглаве рассмотрены все остальные операции, общая суть которых сводится к анализу дерева.

2.1. Базовые операции

В данной главе рассмотрено выполнение основных операций для узлов различных структур деревьев, таких, как вставка, удаление, блокирование,

вставка, перемещение узла в определенное место. В ходе курсовой работы для каждой структуры было выполнено 10 тестов по выполнению данных функций для дерева, содержащего в себе 3000 узлов. Результаты работы описаны в нижеприведенной таблице.

Таблица 2.1: Результаты выполнения программного кода для базовых фукнций

	Список смежности	Перечисление путей	Вложенные множества	Таблица замыканий
Создание нового дерева	0.0768 с.	0.0998 с.	0.0786 с.	0.4933 с
Вставка элемента в середину дерева	0.01944 с.	0.00929 с.	0.04592 с.	0.01304 с
Удаление элемента из середины дерева	0.00637 с.	0.00583 с.	0.0597 с.	0.02378 с
Блокирование элемента дерева	0.00876 с.	0.00745 с.	0.00569 с.	0.02703 с
Перемещение элемента дерева	0.01013 с.	0.01824 с.	0.04727 с.	0.12563 с

Рассмотрим более подробно каждую из этих операций.

2.1.1. Создание нового дерева

Данная операция реализуется через функцию create($username), где $username - имя корневого узла дерева. Чтобы применить данную операцию, нужно сначало удалить предыдущее дерево при его наличии с помощью MySQL-запроса:
DELETE FROM 'имя таблицы',
затем задать значение автоинкрементного поля записи как 1:
ALTER TABLE 'имя таблицы' AUTO_INCREMENT=1,
а после вставить элемент с опредленным именем, который передается функции при вызове:
INSERT INTO 'имя таблицы' VALUES(NULL,NULL, :name).

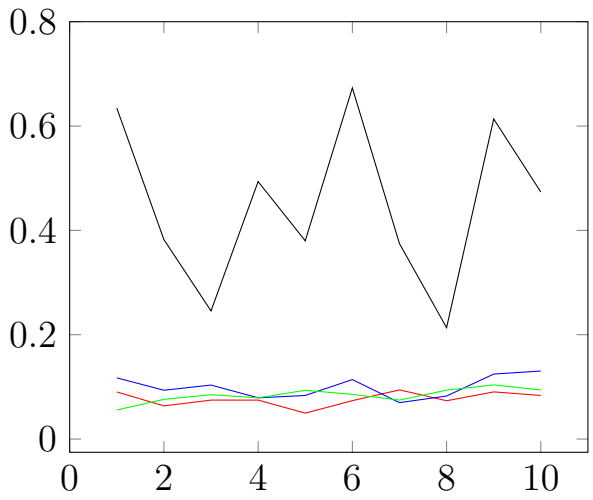

Рис. 2.1: График работы функции create

Данная операция является одной из простейших для дерева, поэтому она с легкостью и быстротой выполняется для всех четырех методов. Методы "Вложенные множества "Перечисление путей"и "Список смежности"реализуют данную операцию гораздо быстрее метода "Таблица замыканий поскольку те содержат в себе одну таблицу, а не две. Было произведено 10 тестов, в результате которых было выведено среднее время работы. На графике 2.1 показано время прохождения каждого теста. Для всех графиков, отражающих время работы функций, примем следующее обозначение цветов графика:

1. красный цвет - структура "Список смежности";
2. синий цвет - структура "Перечисление путей";
3. зеленый цвет - структура "Вложенные множества";
4. черный цвет - структура "Таблица замыканий".

Исходя из результатов работы функции, функция create наиболее быстро работает в однотабличных структурах баз данных "Список смежности "Перечисление путей"и "Вложенные множества". Это свзяано с тем, что функция выполняет запросы к одной таблице базы данных, а не к двум,как в случае с решением "Таблица замыканий".

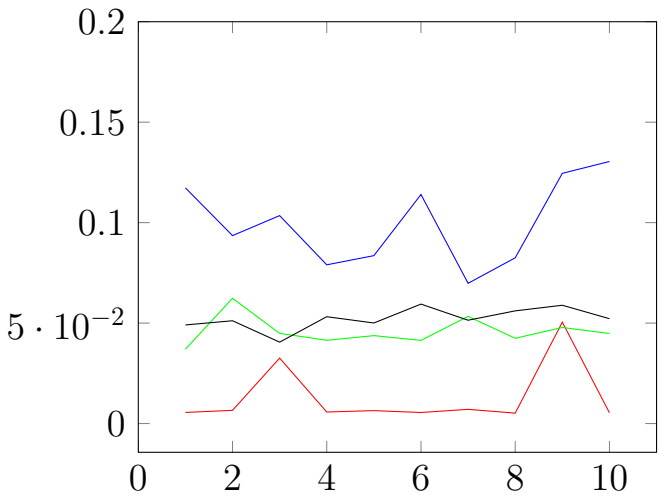

Рис. 2.2: График работы функции create

2.1.2. Вставка элемента в дерево

Вставка узла в дерево-простейшая операция для работы с деревьями, суть которой состоит в добавлении записи в таблицу базы данных, имеющую связь с определенным узлом. Реализуется эта операция через функцию add($id, $user_name), где $user_name - имя вставляемого узла дерева,$id- номер узла дерева, чьим родителем и будет присоединяемый узел. Данная операция выполняется с помощью единственного запроса:

INSERT INTO 'имя таблицы' SET user_name = 'имя пользовтаеля';

Результаты работы данного метода для вставки одного узла в середину дерева из 3000 элементов также, как и при создании нового дерева, зависят от количества таблиц, которые реализуют данный метод и показывают, что наиболее быстро задача решается в решениях "Список смежности"и "Перечисление путей"в связи с тем, что для их реализации не требуется дополнительных запросов в базу данных, в отличие от решения "Вложенные множества и содержат в себе одну таблицу, а не больше, как в структуре "Таблица замыканий".

2.1.3. Удаление или блокирование узла

Для каждого метода хранения деревьев были реализованы функции удаления и блокирования этого узла. Обе эти функции направлены на то, чтобы исключить данный узел из дерева. Однако метод удаления удаляет

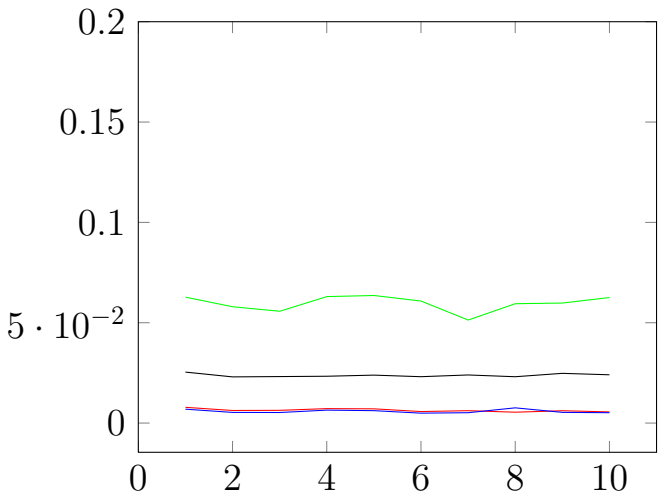

Рис. 2.3: График работы функции delete

узел из дерева безвозвратно, что очень неудобно применять на практике. Для решения этой проблемы в каждую структуру хранения деревьев было добавлено поле user_status, которе хранит информацию о том, заблокирован или нет данный узел. В подобной ситуации логично создать метод блокирования узла, который изменяет лишь поле user_status, отвечающее за состояние элемента, в то время как все остальные данные остаются нетронутыми.

Реализуем данные методы с помощью запросов к базе данных. Так, удаление узла из дерева осуществляется с помощью команды:

DELETE FROM 'имя таблицы' WHERE user_id='id удаляемого узла'

Таким образом, данный элемент удалится из базы данных и восстановить его будет невозможно. Для некторых методов после процесса удаления элемента могут происходить некоторые другие операции, например, в структуре "Вложенные множества"после удаления элемента происходит обновление левых и правых номеров во всей таблице. В программном коде каждого метода этот запрос реализует фукнция delete($id). Рисунок 2.3 показывает, за какое время выполняется программный код этой функции для каждой структуры.

.

В случае же блокирования узла в таблице выполняется команда обновления статуса выбранной записи:

UPDATE 'имя таблицы' SET user_status=0 WHERE user_id='имя бло-

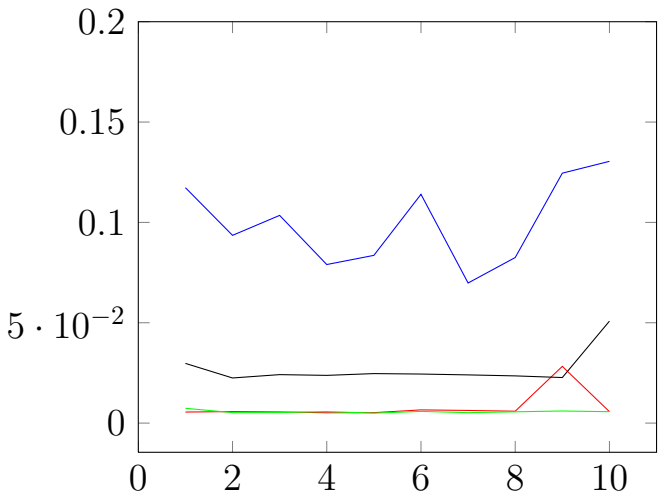

Рис. 2.4: График работы функции block

кируемого узла’

Функция block($id) выполняет данную операцию. Помимо этого, была также реализована функция по разблокировке узла unblock($id), которая по своей сути имеет такой же запрос, как и функция block(), только значение user_status в ней задается равным единице.

Операция block для каждого из реализованных методов выполняется примерно за то же время, что и операция по вставке одного узла в дерево. На рисунке 2.4 показаны результаты тестов для данной функции.

.

2.1.4. Перемещение узла в другое место

Данная операция отвечает за смену родителя для данного узла. Допустим, что некоторый произвольный узел дерева должен быть потомком совершенно другого узла, а не узла, который в настоящее время является родителем. Для того, чтобы переместить узел в нужное место, используется команда move($id, $id_to), где $id- идентификатор перемещаемого узла, $id_to - идентификатор нового родительского узла.

Нельзя не отметить, что при при смене родителя узла меняется расположение не только самого узла, но и всех его дочерних узлов, то есть переносятся все дочерние ветки для данного узла.

Данная операция сводится к выполнению запроса к базе данных, который меняет родителя для данного узла и его детей. С помощью одного

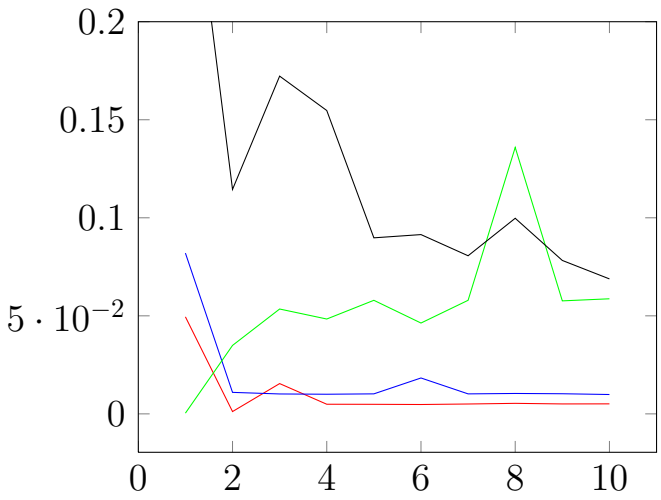

Рис. 2.5: График работы функции move

запроса можно изменить расположение узла дерева в структуре "Список смежности"и "Перечисление путей в которых достаточно изменить расположение одного узла[1]:

UPDATE 'имя таблицы' SET parent_id = $id WHERE user_id= $id_to для структуры "Список смежности

UPDATE 'имя таблицы' SET 'путь' = REPLACE('путь', '$id', '$id_to.') WHERE 'путь' LIKE '$id_to.' для структуры "Перечисление путей".

Для остальных структур для того, чтобы изменить расположение узла, нужно совершить несколько операций. Для структуры "Вложенные множества"нужно сначала обновить левый и правый номера всех узлов,не являющихся данным узлом или его дочерними, а затем вставить в освободившееся место между левым и правым номерами нового родительского узла перемещаемую ветку с уже измененными номерами. В решении "Таблица замыканий"необходимо сначала удалить все записи базы данных с идентификатором $id, а также его потомками, после чего в базу данных вносятся записи, которые показыают связь между данным узлом и его потомками с выбранным узлом и его родителями.

На графике показан результат работы программы при выполнении 10 тестов. Из данного графика и таблицы 2.1 видно, что структура "Список смежности"в данном случае показывает наиболее быстрые результаты. .

2.2. Операции, связанные с анализом дерева

Данные операции позволяют получить всех потомков и родителей данного узла, ветку, где находится данный узел, а также все дерево, где находится данный узел. Нахождение всех узлов, удовлетворяющих данным задачам с помощью одного SQL-запроса, возможно только с помощью методов перечисления путей, списка смежности, а также в некоторых случаях таблицы замыканий.

Для того, чтобы найти с помощью одного SQL-запроса родителей для метода списка смежности, нужно знать точный уровень вложенности данного дерева, что невозможно в реальных условиях. Поэтому для того, чтобы найти потомков(родителей) узла, сначала делают полную выюорку из таблицы базы данных, а затем с помощью рекурсии находят узлы, которые удовлетворяют условию задачи. Данный метод эффективен только при малом объее данных: при большой выборке(более 1000 узлов) преимущество относительного быстродействия утрачивается, выборка может занять свыше 30 секунд.

Таблица 2.2: Результаты выполнения программного кода для функций, анализирующих дерево

	Список смежности	Перечисление путей	Вложенные множества	Таблица замыканий
Потомки узла	19.4234 с.	0.044 с.	0.0052 с.	114.3452 с
Непосредственные потомки узла	0.027 с.	0.02775 с.	0.02736 с.	0.01241 с
Родители узла	0.1234 с.	0.0075 с.	0.0040 с.	0.0129 с
Вывод дерева	30.705 с.	0.04043 с.	0.0306 с.	214.3445 с

2.2.1. Нахождение подчиненной ветки

Функция child_branch($id,$parent_node),где $parent_node принимает значение TRUE,если выборку нужно произвести с учетом узла с идентификатором $id, или FALSE, если выборка проиводится без данного узла, осуществляет вывод всей ветки дерева, для которой корневым узлом является данный узел. С помощью данной функции можно узнть информацию не только о всех узлах, подчиненных данному.

Подчиненную ветку можно легко найти для структур "Вложенные множества"и "Перечисление путей". Для данных методов выполнение функции фактически сводится к запросу в базу данных, использующему сортировку по одному параметру, а также выводу результатов запроса. Запрос в базу данных, который реализует нахождение подчиненной ветки для данных методов, можно представить следующим образом:

SELECT * FROM 'имя таблицы' WHERE user_path LIKE 'путь данного узла' ORDER BY user_path для структуры "Перечисление путей

SELECT user_id, user_name, level FROM 'имя таблицы' WHERE left_number >= 'левый номер' AND right_number <= 'правый номер' ORDER BY left_number для структуры "Вложенные множества".

Однако для структур "Таблица замыканий"и "Список смежности"получение подчиненной ветки одним запросом возможно только до заданного нами уровня подчиненности, поэтому для решения данной задачи применяют рекурсивный алгоритм, неудобный в практическом примнении.

Было произведено 10 опытов по выводу подчиненной ветки первого узла структуры, результаты которого показаны в таблице 2.2. Исходя из результатов этой таблицы, мы подтверждаем наше утверждение о том, что методы, использующие рекурсивные алгоритмы решения данной задачи, не решают на должном уровне данную проблему.

2.2.2. Нахождение непосредственных детей узла

Допустим, что нам нужно найти те комментарии, которые являются ответом на данный комментарий. При этом комментарии, которые являются ответом на дочерние комментарии, не рассматриваются. Такую задачу с легкостью решает алгоритм нахождения непосредственных детей узла. Он реализуется с помощью функции child($id,$parent_node) и находит узлы, которые являются дочерними для данного узла, то есть имеют уровень вложенности на единицу больше, чем запись с идентификатором $id. Эту задачу можно легко решить с помощью всех структур хранения деревьев.

Для решений "Список смежности"и "Таблица замыканий"данная функция выполняется с помощью рекурсии за одну итерацию, что позволяет выполнить данную операцию в рекордно короткие сроки. Решения "Перечисление путей"и "Таблица замыканий"используют в своей сути один запрос к базе данных, после чего происходит вывод значений дерева. Запрос в данных структурах происходит следующим образом:

SELECT * FROM 'имя таблицы' WHERE user_path LIKE 'путь узла' AND user_path NOT LIKE CONCAT('путь узла','/%') ORDER BY user_path для структуры "Перечисление путей

SELECT user_id, user_name, level FROM 'имя таблицы' WHERE left_number >= 'левый номер' AND right_number <= 'правый номер' AND level<='уровень
узла' ORDER BY left_number для структуры "Вложенные множества".

2.2.3. Нахождение родительской ветки для данного узла

Данная основная операция для системы выполняется с помощью функции parent_branch($id,$parent_node). Эта функция выводит список всех родителей для данного узла, а также сам узел, если значение $parent_node равно TRUE. Данная задача решается таким же способом, каким решались два предыдущих метода: для "Списка смежности"и "Таблицы замыканий с помощью запроса и рекурсии, а для "Вложенных множеств"и "Перечисления путей с помощью выполнения запроса и вывода дерева. Следовательно,

для двух последних решений для хранения деревьев запрос по нахождению предков данного узла можно представить так:

SELECT * FROM 'имя таблицы' WHERE 'путь узла' LIKE CONCAT ('путь узла','%') ORDER BY user_path для структуры "Перечисление путей

SELECT user_id, user_name, level FROM 'имя таблицы' WHERE left_number <= 'левый номер' AND right_number >= 'правый номер' ORDER BY left_number для структуры "Вложенные множества".

Резульаты работы функции так же, как и в случае с функцией child_branch, показывают полное превосходство функций, не использующих рекурсию при построении дерева над фукнциями, которые используют ее.

2.2.4. Нахождение полного дерева

Перейдем к нахождению полного дерева все узлов. Данный алгоритм реализует вывод всех узлов дерева, начиная с корневого. Для того, чтобы выполнить вывод всех узлов дерева, нужно выполнить команду tree(). Она выведет все дерево целиком. Для методов "Список смежности"и "Таблица замыканий"вывод дерева осуществляется с помощью рекурсии, а для оставшихся методов-с помощью предварительно отсортированной выборки из базы данных.

В таблице 2.2 представлены реузльтаты по выводу дерева, состоящего из 3000 элементов. Результаты, как и во всех предыдущих методах анализа деревьев, показывают, что методы вложенных множеств и перечисления путей лучше всего справляются с данной задачей. лучше всего справляется с данной операцией.

2.3. Преимущества и недостатки каждой из структур

Изучив все рассмотренные нами структуры, можно сделать вывод об их преимуществах и недостаках[5, 12].

1. Главным достоинством структуры хранения данных "Список смеж-

ности"является ее простота в реализации. В основном разработчики применяют данную структуру для простых операций с деревом, таких, как добавление, удаление или перемещение узла. В то же выполнение операций, которые используют анализ дерева, неэффективно в связи с использованием рекурсивных алгоритмов в программном коде.

2. Структура "Перечисление путей"хорошо подходит для всех основных операций, а также операций, связанных с анализом дерева, но имеет существенный недостаток, связанный с ограничением строки, хранящей в себе путь данного дерева, заключающийся в том, что дерево узлов можно построить только до определенного уровня вложенности. Также данная структура не позволяет установить целостность на уровне ссылок, и как следствие, в структуре приходится хранить избыточные данные об узлах.

3. Структура "Вложенные множества"отлично подходит в тех случаях, когда требуется делать в основном запросы к дереву. Данная структура неудобна для выполнения базовых операций с деревьями и не поддерживает целостность на уровне ссылок.

4. Структура "Таблица замыканий"удобна для выполнения всех операций, кроме получения полной подветки для данного узла, а также полного дерева, которые требуют использования рекурсивных алгоритмов. Структура иммет недостаток, связанный с тем, что занимаемый данной структурой. Это связано с тем, что данная структура использует две таблицы базы данных для хранения информации об узлах дерева. Еще одним недостатком данной структуры является зависимость времени выполнение операции по генерации подветки от количества записей в таблице базы данных: чем больше записей, тем больше времени займет генерация подветки.

Глава 3

Тесты для структур деревьев

Для проверки корректности работы структуры базы данных были разработаны 3 теста: 2-на вставку определенного количества элементов и 1-на обновление статуса элемента. В данной главе описываются алгоритмы вполнения этих тестов, а также показаны результаты работы теста для разных алгоритмов.

3.1. Тест 1. Генерация дочерних узлов дерева.

Допустим, что имеется структура учащихся Бурятского госуниверситета. В нее нужно добавить поступивших на первый курс бакалавриата учащихся наиболее быстрым способом.

Суть задачи заключается в добавлении некоторого проивольного чилса узлов одного уровня к заранее указанному родительскому узлу. Например, суть выше привиденной задачи сводится к добавлению количества узлов, равного числу учащихся в группе, к номеру этой самой группы. В частности, к узлу 05260, отвечающему за номер группы, мы должны добавить 17 дочерних узлов, в которых содержится информация об учащихся данной группы.

Данную задачу можно решить с помощью выполнения в цикле функции add, однако данная реализация будет невыгодной для большого количества узлов. Так, в частности, для вставки 1000 узлов в проивозвольное место может выполняться до 5000 запросов в базу данных в зависимости от метода. Поэтому встает вопрос об использовании такого запроса в базу данных, который поволяет выполнить операцию генерации некоторого количества узлов за как можно меньшее время.

В языке SQL используют фиктивные таблицы DUAL,которые позволяют представить данные из произвольного метода или задачи в виде некоторой таблицы. Эта таблица обычно используется для вставки данных в базу, где они хранятся, или для использования в ситуациях, когда данных о некотором событии в базе данных не имеется. Число строк и столбцов в

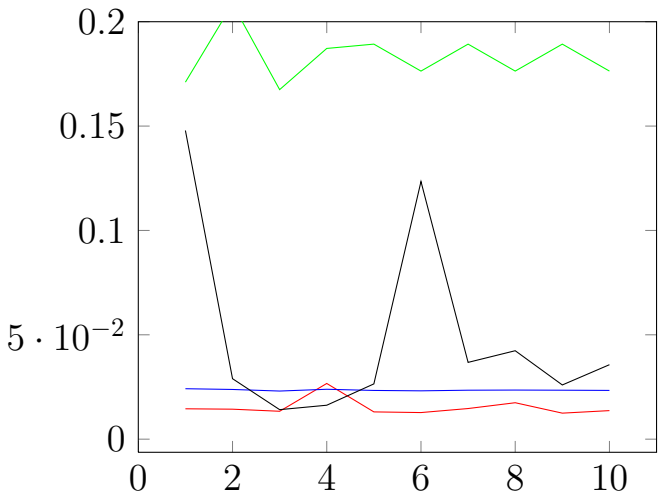

Рис. 3.1: Генерация узлов одного уровня

данной таблице, как и в любой таблице базы данных MySQL,неограничено.

Для решения данной задачи в каждом из решений был реализован запрос вставки в таблицу соответствующей структуры таблицы DUAL, состоящей из всех полей, которые используются в таблице базы данных, реализующей данное решение. К примеру, запрос в базу данных, реализующий вставку двух элементов к некоторому родительскому узлу, выглядить следующим образом :

INSERT INTO 'имя таблицы' SELECT user_id,parent_id,user_name, user_status FROM(SELECT 'id пользователя' user_id, 'id родителя' parent_id,'имя пользователя' user_name,1 user_status FROM DUAL UNION ALL SELECT 'id пользователя_2 ' user_id, 'id родителя' parent_id,'имя пользователя_2' user_name,1 user_status FROM DUAL)t

Реализация данной задачи для третьего уровня вложенности осуществлялась с помощью специально разработанной функции level3($number),где $number- количество вставляемых узлов. Преимуществом данной функции является то, что пользователю не требуется знать информацию о родительском узле. Данные работы функции для вставки 12 произвольных узлов показаны в нижеприведенных рисунке и таблице. .

Таблица 3.1: Результаты выполнения генерации узлов одного уровня

Метод	Количество тестов	Среднее время работы
Список смежности	10	0.0153 с.
Перечисление путей	10	0.023523 с.
Вложенные множества	10	0.18310 с.
Таблица замыканий	10	0.049797 с.

Данные результаты подтверждают наше утверждение о том, что структура "Список смежности"является наиболее оптимальной для вставки элементов в дерево. Также хорошие результаты показывают структуры "Перечисление путей"и "Таблица замыканий". Худший результат показывает решение хранения данных "Вложенные множества"в связи с тем, что перед выполнением операции вставки узлов требуется обновить все узлы в таблице.

3.2. Тест 2. Генерация ветки.

Теперь перед нами стоит более трудная задача. Возьмем, например, систему комментариев. Пользователь хочет прорекламировать свой товар в комментариях, но при этом оставлять рекламу как ответы на комментарии или как отдельный пост. Требуется разработать решение, требующее минимального времени работы.

Данная задача реализуется для каждого метода аналогично предыдущей. Однако эта задача имеет ряд нюансов. В частности, для генерации некоторого количества проивзольных узлов без ограничения вложенности нам требуется знать два параметра: идентификатор родительского узла и количество вставляемых узлов. Они используются в функции multinsert($id_to,$number),где $id_to,$number-это номер родительского узла и количество моделируемых узлов соответственно. В реализации функции на языке PHP задается цикл, в каждой итерации которого генерируется псевдослучайное число в промежутке от числа вставленных строк в таблице базы данных до суммы этого числа и количества узлов, которое требуется смоделировать. Если число равно левой границы отрезка генера-

ции, то родителем для данного узла считается узел, идетификатор которого задан как параметр функции, в противном случае родителем для нового узла считается один из сгенерированных в цикле узлов. После генерации узла информация о нем добавляется в таблицу DUAL. По окончании цикла выполняется запрос к базе данных, котрый выполняет операцию вставки некоторого числа узлов к заданному родительскому узлу. Нижеприведенный запрос выполняет вставку поддерева из двух записей в дерево структуры "Список смежности":

INSERT INTO 'имя таблицы' SELECT user_id,parent_id,user_name, user_status FROM(SELECT 'id пользователя' user_id, 'id родителя' parent_id,'имя пользователя' user_name,1 user_status FROM DUAL UNION ALL SELECT 'id пользователя_2 ' user_id, 'id пользователя' parent_id,'имя пользователя_2' user_name,1 user_status FROM DUAL)t

Было проведено 10 тестов по вставке 1000 узлов в дерево для каждой структуры. Результаты этих тестов показаны в табилце 3.2 и на рисунке 3.2. Они подтверждают, что наиболее быстрой в данной ситуации является структура хранения деревьев "Список смежности а структуры "Вложенные множества"и "Перечисление путей"показывают удовлетворительные результаты. Решение хранения деревьев "Список смежности"с каждым новым тестом показывает время работы, худшее, чем на предшествующем ему тесте. Это связано с тем, что количество данных в таблице базы данных для данного метода увеличивается с каждым новым тестом, и, как следствие, увеличивается время работы с базой данных.

.

Таблица 3.2: Результаты выполнения генерации подветки

Метод	Количество тестов	Среднее время работы
Список смежности	10	0.06634 с.
Перечисление путей	10	0.12517 с.
Вложенные множества	10	0.36756 с.
Таблица замыканий	10	3.872 с.

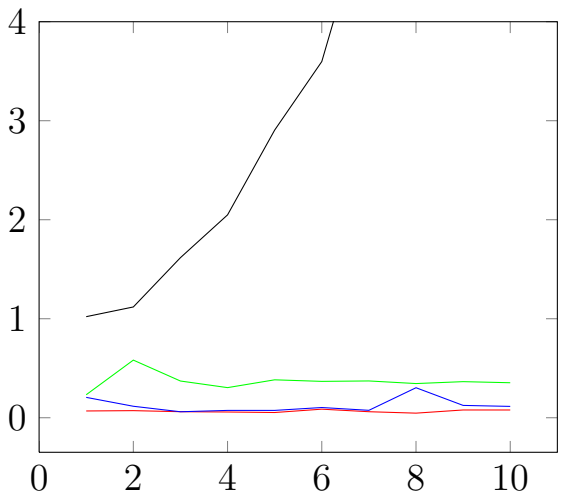

Рис. 3.2: Генерация подветки

3.3. Тест 3. Проход по дочерним узлам.

Допустим, что у нас имеется узел, обладающий определенным свойством. Требуется распространить это свойство на всех его потомков.

Данная задача схожа с одним из реализованных нами методов по нахождению подчиненной ветки узла. Основной алгоритм теста, используемый нами в функции test5($id), схож с реализацией функциия child_branch($id). Однако для данной структуры требуется лишь список узлов дочерней ветки, в связи с чем можно сделать вывод, что данная операция будет показывать быстрое время работы для всех структур, кроме "Списка смежности в котором для получения всех узлов дочерней ветки требуется выполнение рекурсивной функции. В функции, реализующей данный тест, задется массив узлов, в которых нужно изменить свойство. Для структуры "Список смежности"ы на каждой итерации в массив добавляется номер узла, который является подчиненным для заданного узла. Для всех остальных идентификатор подчиненных узлов также заносится в массив. После внесения данных в массив начинается обновление всех выбранных записей с помощью команды:

UPDATE ’имя таблицы’ SET user_status=CASE user_id WHEN ’имя пользователя_1’ THEN ’1’ WHEN ’имя пользователя_2’ THEN ’1’ ELSE ’user_status’ END;

Данная операция реализована для всех структур хранения деревьев.

Таблица 3.3 показывает результаты работы при задании идентификатора узла дерева, равного 1. В результате данного теста наиболее эффективными является структуры "Вложенные множества"и "Табилца замыканий не использующие в решении данной задачи рекурсию.

Таблица 3.3: Результаты выполнения генерации подветки

Метод	Количество тестов	Среднее время работы
Список смежности	10	31.62355 с.
Перечисление путей	10	0.118968 с.
Вложенные множества	10	0.1129879 с.
Таблица замыканий	10	0.158152 с.

ЗАКЛЮЧЕНИЕ

В результате курсовой работы были получены следующие результаты:

1. Рассмотрены четыре основные структуры хранения деревьев в базе данных.
2. Изучены их преимущества и недостатки.
3. Был реализован и протестирован программный код данных структур для конкретных методов.

Рассмотренная в рамках курсовой работы тема имеет важную практическую ценность. С помощью деревьев можно реализовать множество реальных практических задач, таких, как получение списка друзей и подпичсиков для каждого пользователя социальной сети, дисконтно-бонусные программы магазинов, системы комментариев для сайта и т.д. В настоящее время в реализации наиболее широко используют четыре основные структуры хранения деревьев- "Список смежности "Перечисление путей "Вложенные множества "Табилца замыканий". Несмотря на все свои недостатки, каждая из данных структур широко используется на практике в различных задачах не только как отдельные решения для хранения деревьев, но и в симбиозе друг с другом для получения более оптимальных результатов работы с большими данными.

Программный код каждой из четырех структур хранения деревьев доступен в репозитории, находящемся в открытом доступе в сети Интернет по адресу: https://github.com/komantnick/tree-structures-php . В дальнейшем планируется разработка модуля на основе PHP-фреймворка Yii2 для работы со структурами данного класса, а также улучшение программного кода для вывода деревьев с помощью рекурсивного алгоритма и возможная его замена на нерекурсивный.

Литература

1. Веллинг Л. MySQL. Учебное пособие. - Л. Веллинг, Л. Томсон. - М.: ООО «И.Д. Вильямс», 2005, – 304 с.

2. Грошев, А. С. Базы данных: Учебное пособие. - А. С. Грошев. - Архангельск, Изд-во Арханг. гос. тех. ун-та, 2005. - 124 с.

3. Ермаков, Д. Г. Моделирование иерархических объектов средствами реляционных СУБД. - Д. Г. Ермаков. - Екатеринбург: УГТУ-УПИ, 2007 г. - 132 с.

4. Зандстра М. PHP: объекты, шаблоны и методы программирования, 3-е изд.// М. Зандстра. - М.: ООО «И.Д. Вильямс», 2011, – 560 с.

5. Карвин, Б. Программирование баз данных SQL. Типичные ошибки и их устранение. / Б. Карвин. - М. : Рид Групп, 2012. - 336 с.

6. Кормен, Томас Х. и др. Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. - М.: ООО «И.Д. Вильямс», 2013, – 1328 с.

7. Руководство по MySQL [электронный ресурс] режим доступа: http://dev.mysql.com/

8. Руководство по PHP [электронный ресурс] режим доступа: http://php.net/

9. Тарасов, С. В. СУБД для программиста. Базы данных изнутри. - С. В. Тарасов. - М.: СОЛОН-Пресс, 2015. - 320 с.

10. Шварц, Б. и др. MySQL. Оптимизация проивзодительности, 2-е издание. : Пер. с англ. - СПб.: Символ-Плюс, 2010, – 832 с.

11. Celco, J. Joe Celco's Trees and Hierarchies in SQL for Smarties. / J. Celco. - Morgan Kaufmann, 2004. - 224 p.

12. Van Tulder, Gijs. Storing Hierarchical Data in a Database [электронный ресурс] режим доступа: https://www.sitepoint.com/hierarchical-data-database/