

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Департамент программной инженерии

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»,
профессор департамента программной
инженерии, канд. техн. наук

«__» _____ 2019 г. В.В. Шилов

Выпускная квалификационная работа

на тему «Компонент-расширение РСУБД SQLite для индексирования данных
модификациями В-деревьев»

по направлению подготовки 09.03.04 «Программная инженерия»

ПРИЛОЖЕНИЯ

Научный руководитель
Старший преподаватель
департамента программной
инженерии

С.А. Шершаков

Подпись, Дата

Выполнил
студент группы БПИ153
4 курса бакалавриата
образовательной программы
«Программная инженерия»

А.М. Ригин

Подпись, Дата

Москва 2019

Содержание

Приложение А. Техническое задание.....	36
Приложение Б. Программа и методика испытаний	52
Приложение В. Руководство оператора	77
Приложение Г. Текст программы	89

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО

Старший преподаватель департамента
программной инженерии факультета
компьютерных наук

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ С.А. Шершаков
«__» _____ 2019 г.

_____ В.В. Шилов
«__» _____ 2019 г.

КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ

Техническое задание

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.02.09-01 ТЗ 01-1-ЛУ

Исполнитель
студент группы БПИ153
_____ / Ригин А. М. /
«__» _____ 2019 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.02.09-01 ТЗ 01-1-ЛУ

Москва 2019

УТВЕРЖДЕН
RU.17701729.02.09-01 ТЗ 01-1-ЛУ

**КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ**

Техническое задание

RU.17701729.02.09-01 ТЗ 01-1

Листов 15

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.02.09-01 ТЗ 01-1				

Москва 2019

Содержание

1. Введение	39
2. Основания для разработки	40
3. Назначение разработки	41
3.1. Функциональное назначение	41
3.2. Эксплуатационное назначение	41
4. Требования к программе	42
4.1. Требования к функциональным характеристикам	42
4.1.1. Требования к составу выполняемых функций	42
4.1.2. Требования к организации входных данных	43
4.1.3. Требования к организации выходных данных	44
4.2. Требования к надежности	44
4.3. Требования к интерфейсу	44
4.4. Условия эксплуатации.....	44
4.5. Требования к составу и параметру технических средств	44
4.6. Требования к информационной и программной совместимости	44
4.7. Требования к маркировке и упаковке	45
4.8. Требования к транспортированию и хранению	45
5. Требования к программной документации.....	46
6. Техничко-экономические показатели	47
6.1. Предполагаемая потребность	47
6.2. Ориентировочная экономическая эффективность.....	47
6.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами	47
7. Стадии и этапы разработки.....	48
7.1. Необходимые стадии разработки, этапы и содержание работ	48
7.2. Сроки разработки и исполнители.....	50
8. Порядок контроля и приемки	51
8.1. Виды испытаний	51
8.2. Общие требования к приемке работы.....	51

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

1. Введение

Наименование программы: «Компонент-расширение РСУБД SQLite для индексирования данных модификациями В-деревьев».

Программа будет применяться для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, а также вывода графического представления используемого в качестве индексирующей структуры данных В-дерева или его модификации в DOT-файл для GraphViz и основных данных об используемом дереве.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2. Основания для разработки

Основанием для разработки программы является Приказ НИУ ВШЭ № 2.3-02/1012-01 от 10.12.2018 г.

Программа разрабатывается в рамках выполнения выпускной квалификационной работы на тему «Компонент-расширение РСУБД SQLite для индексирования данных модификациями В-деревьев».

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

3. Назначение разработки

3.1. Функциональное назначение

Программа будет применяться для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, как расширение для РСУБД SQLite, позволяющее работать с таблицами, созданными при помощи данного расширения, и выводить графическое представление В-дерева или его модификации, используемой в данной таблице, в DOT-файл для GraphViz, и основные данные, связанные с соответствующей индексирующей структурой данных (деревом).

3.2. Эксплуатационное назначение

Программа будет применяться разработчиками и исследователями для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, а также вывода графического представления используемого в качестве индексирующей структуры данных В-дерева или его модификации в DOT-файл для GraphViz и основных данных об используемом дереве, в том числе, в учебных и научных целях.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

4. Требования к программе

4.1. Требования к функциональным характеристикам

4.1.1. Требования к составу выполняемых функций

Расширение для SQLite (программа) должно удовлетворять следующим функциональным требованиям:

1. Расширение должно позволять создавать таблицу, использующую В⁺-дерево из данного расширения в качестве индексирующей структуры данных, с указанием столбца, являющегося первичным ключом таблицы.
2. Расширение должно позволять удалять таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
3. Расширение должно позволять производить поиск строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
4. Расширение должно позволять производить вставку строки/строк в таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
5. Расширение должно позволять производить удаление строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
6. Расширение должно позволять производить обновление значений ячеек (включая ячейку с первичным ключом) строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

7. Расширение должно позволять переименовывать таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
8. Расширение должно при каждой операции с таблицей, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, запускать алгоритм выбора индексирующей структуры данных из модификаций В-дерева (B^+ -дерева, B^* -дерева и B^{*+} -дерева) и перестраивать имеющуюся индексирующую структуру данных на новую (если была выбрана новая), сохраняя все имеющиеся в ней данные.
9. Расширение должно поддерживать сохранение таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вместе с базой данных на постоянном запоминающем устройстве.
10. Расширение должно поддерживать открытие сохранённой вместе с базой данных на постоянном запоминающем устройстве таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
11. Расширение должно поддерживать для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод графического представления индексирующей структуры данных (дерева) таблицы в DOT-файл для GraphViz.
12. Расширение должно поддерживать для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод типа используемого дерева (1 — В-дерево, 2 — B^+ -дерево, 3 — B^* -дерево, 4 — B^{*+} -дерево).
13. Расширение должно поддерживать для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод порядка используемого дерева.

4.1.2. Требования к организации входных данных

Программа должна позволять вводить входные данные (запросы к базе данных) через командную строку.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

4.1.3. Требования к организации выходных данных

Программа должна позволять выводить выходные данные (ответы базы данных) через командную строку.

4.2. Требования к надежности

Программа обеспечивает проверку корректности входных данных.

Для корректной работы программы требуется стабильное и корректное функционирование компьютера и операционной системы.

4.3. Требования к интерфейсу

Программа должна иметь интерфейс командной строки с возможностью ввода входных данных (запросов к базе данных) и вывода выходных данных (ответов базы данных) в командной строке.

4.4. Условия эксплуатации

Требуемая квалификация пользователя программы — оператор ЭВМ с базовыми знаниями в области работы с системами управления базами данных (СУБД).

4.5. Требования к составу и параметру технических средств

Для нормального функционирования программы требуется компьютер, оснащенный следующими техническими компонентами:

- 1) процессор не ниже Intel Pentium/Celeron, AMD K6/Athlon/Duron или совместимый с ними с тактовой частотой не ниже 1 ГГц;
- 2) 512 Мб ОЗУ или более;
- 3) жесткий диск с объемом свободной памяти не менее 100 Мб;
- 4) VGA-совместимые видеоадаптер и монитор;
- 5) клавиатура.

4.6. Требования к информационной и программной совместимости

Для нормального функционирования программы требуется компьютер, оснащенный следующими программными компонентами:

- 1) операционная система Microsoft Windows 7 / 8 / 8.1 / 10 либо Ubuntu версии 16.04 LTS или выше;

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2) SQLite 3 версии 3.24 или выше.

4.7. Требования к маркировке и упаковке

Программа поставляется пользователю на электронном носителе информации в виде динамической библиотеки — файл с расширением .dll (для Windows) либо .so (для Linux).

Программа сразу готова к запуску, её установка не требуется.

4.8. Требования к транспортированию и хранению

Требования к транспортированию и хранению программы соответствуют стандартным требованиям к транспортированию и хранению соответствующих электронных и бумажных носителей информации.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

5. Требования к программной документации

Состав программной документации должен включать в себя следующие компоненты:

- 1) Техническое задание (ГОСТ 19.201-78)
- 2) Программа и методика испытаний (ГОСТ 19.301-78)
- 3) Руководство оператора (ГОСТ 19.505-79)
- 4) Текст программы (ГОСТ 19.401-78)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

6. Техничко-экономические показатели

6.1. Предполагаемая потребность

Программа будет применяться разработчиками и исследователями для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, а также вывода графического представления используемого в качестве индексирующей структуры данных В-дерева или его модификации в DOT-файл для GraphViz и основных данных об используемом дереве, в том числе, в учебных и научных целях.

6.2. Ориентировочная экономическая эффективность

Программа сможет бесплатно расширить функционал SQLite, что может быть использовано разработчиками и исследователями в области СУБД, алгоритмов и структур данных.

6.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами

Аналогов программы в открытом доступе не обнаружено.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

7. Стадии и этапы разработки

7.1. Необходимые стадии разработки, этапы и содержание работ

Стадии разработки, этапы и содержание работ составлены в соответствии с ГОСТ 19.102–77.

I. Техническое задание

1. Обоснование необходимости разработки программы

- 1) постановка задачи;
- 2) сбор исходных материалов;
- 3) выбор и обоснование критериев эффективности и качества разрабатываемой программы;
- 4) обоснование необходимости проведения научно-исследовательских работ.

2. Научно-исследовательские работы

- 1) определение структуры входных и выходных данных;
- 2) предварительный выбор методов решения задач;
- 3) обоснование целесообразности применения ранее разработанных программ;
- 4) определение требований к техническим средствам;
- 5) обоснование принципиальной возможности решения поставленной задачи.

3. Разработка и утверждение технического задания

- 1) определение требований к программе;
- 2) определение стадий, этапов и сроков разработки программы и документации на нее;
- 3) выбор языков программирования;
- 4) определение необходимости проведения научно-исследовательских работ на последующих стадиях;
- 5) согласование и утверждение технического задания.

II. Эскизный проект

1. Разработка эскизного проекта

- 1) предварительная разработка структуры входных и выходных данных;
- 2) уточнение методов решения задачи;

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

3) разработка общего описания алгоритма решения задачи;

2. Утверждение эскизного проекта

1) разработка пояснительной записки;

2) согласование и утверждение эскизного проекта.

III. Технический проект

1. Разработка технического проекта

1) уточнение структуры входных и выходных данных;

2) разработка алгоритма решения задачи;

3) определение формы представления входных и выходных данных;

4) разработка структуры программы;

5) окончательное определение конфигурации технических средств.

2. Утверждение технического проекта

1) разработка плана мероприятий по разработке программы;

2) разработка пояснительной записки;

3) согласование и утверждение технического проекта.

IV. Рабочий проект

1. Разработка программы

1) программирование и отладка программы.

2. Разработка программной документации

1) разработка программных документов в соответствии с требованиями ГОСТ 19.101–77.

3. Испытания программы

1) разработка, согласование и утверждение программы и методики испытаний;

2) проведение предварительных испытаний;

3) корректировка программы и программной документации по результатам испытаний.

V. Внедрение

1. Подготовка и передача программы

1) подготовка и передача программы и программной документации для сопровождения и (или) изготовления;

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2) защита программы комиссии по защите выпускной квалификационной работы.

7.2. Сроки разработки и исполнители

Программа и документация к ней разрабатываются к утвержденным срокам сдачи выпускной квалификационной работы (не позднее 28 мая 2019 года).

Исполнителем является студент НИУ ВШЭ группы БПИ153 Ригин Антон Михайлович.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

8. Порядок контроля и приемки

8.1. Виды испытаний

Виды испытаний описаны в документе «Программа и методика испытаний» (ГОСТ 19.301-78).

8.2. Общие требования к приемке работы

Общие требования к приемке работы описаны в документе «Программа и методика испытаний» (ГОСТ 19.301-78).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО

Старший преподаватель департамента
программной инженерии факультета
компьютерных наук

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ С.А. Шершаков
«__» _____ 2019 г.

_____ В.В. Шилов
«__» _____ 2019 г.

КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ

Программа и методика испытаний

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.02.09-01 51 01-1-ЛУ

Исполнитель
студент группы БПИ153
_____ / Ригин А. М. /
«__» _____ 2019 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.02.09-01 51 01-1-ЛУ

Москва 2019

УТВЕРЖДЕН
RU.17701729.02.09-01 51 01-1-ЛУ

**КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ**

Программа и методика испытаний

RU.17701729.02.09-01 51 01-1

Листов 24

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.02.09-01 51 01-1				

Москва 2019

Содержание

1. Объект испытаний	56
1.1. Наименование программы	56
1.2. Область применения.....	56
1.3. Обозначение испытуемой программы	56
2. Цель испытаний.....	57
3. Требования к программе	58
3.1. Требования к функциональным характеристикам	58
3.1.1. Требования к составу выполняемых функций	58
3.1.2. Требования к организации входных данных	59
3.1.3. Требования к организации выходных данных	60
3.2. Требования к надежности	60
3.3. Требования к интерфейсу	60
4. Требования к программной документации.....	61
5. Средства и порядок испытаний.....	62
5.1. Технические средства, используемые во время испытаний	62
5.2. Программные средства, используемые во время испытаний	62
5.3. Порядок проведения испытаний	62
5.4. Условия проведения испытаний.....	62
5.4.1. Климатические условия.....	62
5.4.2. Требования к численности и квалификации персонала	63
6. Методы испытаний	64
6.1. Испытание выполнения требований к программной документации.....	64
6.2. Испытание выполнения требований к интерфейсу	64
6.3. Испытание выполнения требований к функциональным характеристикам	65
6.3.1. Испытание выполнения требований к функциональным характеристикам в части создания таблицы.....	65
6.3.2. Испытание выполнения требований к функциональным характеристикам в части удаления таблицы	65
6.3.3. Испытание выполнения требований к функциональным характеристикам в части поиска строки/строка в таблице	65
6.3.4. Испытание выполнения требований к функциональным характеристикам в части вставки строки/строка в таблицу	66

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

6.3.5. Испытание выполнения требований к функциональным характеристикам в части удаления строки/строк в таблице	66
6.3.6. Испытание выполнения требований к функциональным характеристикам в части обновления значений ячеек (включая ячейку с первичным ключом) строки/строк в таблице	67
6.3.7. Испытание выполнения требований к функциональным характеристикам в части переименования таблицы	69
6.3.8. Испытание выполнения требований к функциональным характеристикам в части алгоритма выбора индексирующей структуры данных и перестраивания дерева	70
6.3.9. Испытание выполнения требований к функциональным характеристикам в части сохранения таблицы, вместе с базой данных, на постоянном запоминающем устройстве, и открытия такой сохранённой таблицы	72
6.3.10. Испытание выполнения требований к функциональным характеристикам в части вывода графического изображения индексирующей структуры данных (дерева) таблицы в DOT-файл для GraphViz	73
6.3.11. Испытание выполнения требований к функциональным характеристикам в части вывода типа используемого в таблице дерева	74
6.3.12. Испытание выполнения требований к функциональным характеристикам в части вывода порядка используемого в таблице дерева	75
6.4. Испытание выполнения требований к надёжности	76

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

1. Объект испытаний

1.1. Наименование программы

Наименование программы: «Компонент-расширение РСУБД SQLite для индексирования данных модификациями В-деревьев».

1.2. Область применения

Программа будет применяться для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, а также вывода графического представления используемого в качестве индексирующей структуры данных В-дерева или его модификации в DOT-файл для GraphViz и основных данных об используемом дереве.

1.3. Обозначение испытываемой программы

Краткое наименование программы — btrees_mods.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2. Цель испытаний

Цель испытаний — проверка соответствия функционала и характеристик программного продукта требованиям к программному продукту, изложенным в документе «Техническое задание» (ГОСТ 19.201-78).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

3. Требования к программе

3.1. Требования к функциональным характеристикам

3.1.1. Требования к составу выполняемых функций

Расширение для SQLite (программа) должно удовлетворять следующим функциональным требованиям:

1. Расширение должно позволять создавать таблицу, использующую В⁺-дерево из данного расширения в качестве индексирующей структуры данных, с указанием столбца, являющегося первичным ключом таблицы.
2. Расширение должно позволять удалять таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
3. Расширение должно позволять производить поиск строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
4. Расширение должно позволять производить вставку строки/строк в таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
5. Расширение должно позволять производить удаление строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
6. Расширение должно позволять производить обновление значений ячеек (включая ячейку с первичным ключом) строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

7. Расширение должно позволять переименовывать таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
8. Расширение должно при каждой операции с таблицей, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, запускать алгоритм выбора индексирующей структуры данных из модификаций В-дерева (B^+ -дерева, B^* -дерева и B^{*+} -дерева) и перестраивать имеющуюся индексирующую структуру данных на новую (если была выбрана новая), сохраняя все имеющиеся в ней данные.
9. Расширение должно поддерживать сохранение таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вместе с базой данных на постоянном запоминающем устройстве.
10. Расширение должно поддерживать открытие сохранённой вместе с базой данных на постоянном запоминающем устройстве таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
11. Расширение должно поддерживать для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод графического представления индексирующей структуры данных (дерева) таблицы в DOT-файл для GraphViz.
12. Расширение должно поддерживать для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод типа используемого дерева (1 — В-дерево, 2 — B^+ -дерево, 3 — B^* -дерево, 4 — B^{*+} -дерево).
13. Расширение должно поддерживать для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод порядка используемого дерева.

3.1.2. Требования к организации входных данных

Программа должна позволять вводить входные данные (запросы к базе данных) через командную строку.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

3.1.3. Требования к организации выходных данных

Программа должна позволять выводить выходные данные (ответы базы данных) через командную строку.

3.2. Требования к надежности

Программа обеспечивает проверку корректности входных данных.

Для корректной работы программы требуется стабильное и корректное функционирование компьютера и операционной системы.

3.3. Требования к интерфейсу

Программа должна иметь интерфейс командной строки с возможностью ввода входных данных (запросов к базе данных) и вывода выходных данных (ответов базы данных) в командной строке.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

4. Требования к программной документации

Состав программной документации должен включать в себя следующие компоненты:

- 1) Техническое задание (ГОСТ 19.201-78)
- 2) Программа и методика испытаний (ГОСТ 19.301-78)
- 3) Руководство оператора (ГОСТ 19.505-79)
- 4) Текст программы (ГОСТ 19.401-78)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

5. Средства и порядок испытаний

5.1. Технические средства, используемые во время испытаний

Для нормального функционирования программы требуется компьютер, оснащенный следующими техническими компонентами:

- 1) процессор не ниже Intel Pentium/Celeron, AMD K6/Athlon/Duron или совместимый с ними с тактовой частотой не ниже 1 ГГц;
- 2) 512 Мб ОЗУ или более;
- 3) жесткий диск с объемом свободной памяти не менее 100 Мб;
- 4) VGA-совместимые видеоадаптер и монитор;
- 5) клавиатура.

5.2. Программные средства, используемые во время испытаний

Для нормального функционирования программы требуется компьютер, оснащенный следующими программными компонентами:

- 1) операционная система Microsoft Windows 7 / 8 / 8.1 / 10 либо Ubuntu версии 16.04 LTS или выше;
- 2) SQLite 3 версии 3.24 или выше.

5.3. Порядок проведения испытаний

Испытания должны проводиться в следующем порядке:

- 1) проверка требований к программной документации;
- 2) проверка требований к интерфейсу;
- 3) проверка требований к функциональным характеристикам;
- 4) проверка требований к надежности.

5.4. Условия проведения испытаний

5.4.1. Климатические условия

Климатические условия проведения испытаний программного продукта должны удовлетворять стандартным требованиям к климатическим условиям использования компьютера и использования и хранения соответствующих электронных и бумажных носителей информации.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

5.4.2. Требования к численности и квалификации персонала

Для испытаний программы требуется по крайней мере один пользователь.

Требуемая квалификация пользователя программы — оператор ЭВМ с базовыми знаниями в области работы с системами управления базами данных (СУБД).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

6. Методы испытаний

Испытания проводятся в порядке, указанном в п. 5.3 настоящего документа.

6.1. Испытание выполнения требований к программной документации

Соответствие программной документации требованиям проверяется путем просмотра программной документации вручную.

Путем просмотра выявлено, что программная документация удовлетворяет требованиям.

6.2. Испытание выполнения требований к интерфейсу

Программа имеет интерфейс командной строки с возможностью ввода входных данных (запросов к базе данных) и вывода выходных данных (ответов базы данных) в командной строке, как показано на скриншоте на рис. 1, поскольку использует интерфейс командной строки РСУБД SQLite.

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods(id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> INSERT INTO btt VALUES (4, 2, "ABC123");
sqlite> INSERT INTO btt VALUES (7, 3, "def");
sqlite> SELECT * FROM btt WHERE id = 4;
4|2|ABC123
sqlite> SELECT * FROM btt WHERE id = 7;
7|3|def
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|ABC123
7|3|def
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btt_real;
4|2|ABC123
7|3|def
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|2|0|id|INTEGER|4|tree_33051558297088.btree
sqlite> DROP TABLE btt;
sqlite> .tables
btrees_mods_idxinfo
sqlite> SELECT * FROM btrees_mods_idxinfo;
sqlite> .exit
```

Рисунок 1. Интерфейс РСУБД SQLite с подключённым расширением btrees_mods

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

Таким образом, программа полностью соответствует требованиям к интерфейсу.

6.3. Испытание выполнения требований к функциональным характеристикам

6.3.1. Испытание выполнения требований к функциональным характеристикам в части создания таблицы

Испытание выполнения требований к функциональным характеристикам в части создания таблицы, использующей В⁺-дерево из данного расширения в качестве индексирующей структуры данных, с указанием столбца, являющегося первичным ключом таблицы выполнено в п. 6.2 настоящего документа, где на скриншоте на рис. 1 показано, что таблица, использующая В⁺-дерево из данного расширения в качестве индексирующей структуры данных, успешно создана, и, таким образом, программа удовлетворяет данному функциональному требованию.

6.3.2. Испытание выполнения требований к функциональным характеристикам в части удаления таблицы

Испытание выполнения требований к функциональным характеристикам в части удаления таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, выполнено в п. 6.2 настоящего документа, где на скриншоте на рис. 1 показано, что таблица, использующая В⁺-дерево из данного расширения в качестве индексирующей структуры данных, успешно удалена, и, таким образом, программа удовлетворяет данному функциональному требованию.

6.3.3. Испытание выполнения требований к функциональным характеристикам в части поиска строки/строк в таблице

Испытание выполнения требований к функциональным характеристикам в части поиска строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям, выполнено в п. 6.2 настоящего документа, где на скриншоте на рис. 1 показано, что поиск как одной строки, так и нескольких строк, в

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, успешно произведён, и, таким образом, программа удовлетворяет данному функциональному требованию.

6.3.4. Испытание выполнения требований к функциональным характеристикам в части вставки строки/строк в таблицу

Испытание выполнения требований к функциональным характеристикам в части вставки строки/строк в таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, частично выполнено в п. 6.2 настоящего документа, где на скриншоте на рис. 1 показано, что вставка одной строки в таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, успешно произведена.

Вставим несколько строк в таблицу, как показано на скриншоте на рис. 2.

```
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods(id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> INSERT INTO btt VALUES (4, 2, "ABC123"), (7, 3, "def");
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|ABC123
7|3|def
sqlite>
```

Рисунок 2. Вставка нескольких строк в таблицу

Как мы видим, вставка нескольких строк в таблицу успешно произведена. Таким образом, программа удовлетворяет данному функциональному требованию.

6.3.5. Испытание выполнения требований к функциональным характеристикам в части удаления строки/строк в таблице

Проведём испытание выполнения требований к функциональным характеристикам в части удаления строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям. Продолжим работу с таблицей, созданной в рамках испытаний в п. 6.3.3. (см. скриншот на рис. 2). Удалим строку из таблицы, как показано на скриншоте на рис. 3. Как мы видим, строка успешно удалена.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```
sqlite> DELETE FROM btt WHERE id = 7;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|ABC123
sqlite>
```

Рисунок 3. Удаление строки из таблицы

Проверим удаление нескольких строк из таблицы. Для этого предварительно вставим в таблицу ещё одну строку. Данные действия показаны на скриншоте на рис. 4.

```
sqlite> INSERT INTO btt VALUES (8, 727, "SEFJKWLEJWLKEJD");
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 8;
4|2|ABC123
8|727|SEFJKWLEJWLKEJD
sqlite> DELETE FROM btt WHERE id = 4 OR id = 8;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 8;
sqlite>
```

Рисунок 4. Удаление нескольких строк из таблицы

Как мы видим, удаление нескольких строк из таблицы успешно произведено. Таким образом, программа удовлетворяет данному функциональному требованию.

6.3.6. Испытание выполнения требований к функциональным характеристикам в части обновления значений ячеек (включая ячейку с первичным ключом) строки/строк в таблице

Проведём испытание выполнения требований к функциональным характеристикам в части обновления значений ячеек (включая ячейку с первичным ключом) строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.

Создадим таблицу и вставим в неё 4 элемента, как показано на скриншоте на рис. 5.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods(id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> INSERT INTO btt VALUES (4, 2, "ABC123"), (7, 3, "def"), (5, 4, "456"), (6, 1, "LKJLKJ");
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7 OR id = 5 OR id = 6;
4|2|ABC123
5|4|456
6|1|LKJLKJ
7|3|def

```

Рисунок 5. Создание таблицы с 4 элементами

Далее обновим значение первичного ключа у одной из строк, как показано на скриншоте на рис. 6.

```

sqlite> UPDATE btt SET id = 10 WHERE id = 6;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7 OR id = 5 OR id = 6 OR id = 10;
4|2|ABC123
5|4|456
7|3|def
10|1|LKJLKJ

```

Рисунок 6. Обновление первичного ключа строки таблицы

Как мы видим, данное действие успешно выполнено. Теперь обновим значение ячейки, не являющейся первичным ключом, у одной из строк таблицы, как показано на скриншоте на рис. 7.

```

sqlite> UPDATE btt SET a = 7 WHERE id = 7;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7 OR id = 5 OR id = 10;
4|2|ABC123
5|4|456
7|7|def
10|1|LKJLKJ

```

Рисунок 7. Обновление ячейки, не являющейся первичным ключом, у одной из строк таблицы

Как мы видим, данное действие также выполнено успешно. Наконец, обновим значение ячеек столбца, не являющегося первичным ключом, у нескольких строк таблицы, как показано на скриншоте на рис. 8.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

sqlite> UPDATE btt SET b = "ok" WHERE id = 4 OR id = 5;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7 OR id = 5 OR id = 10;
4|2|ok
5|4|ok
7|7|def
10|1|LKJLKJ

```

Рисунок 8. Обновление значения ячеек столбца, не являющегося первичным ключом, у нескольких строк таблицы

Как мы видим, данное действие также выполнено успешно. Таким образом, программа удовлетворяет данному функциональному требованию.

6.3.7. Испытание выполнения требований к функциональным характеристикам в части переименования таблицы

Создадим и переименуем таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, как показано на скриншоте на рис. 9.

```

SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods(id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|2|0|id|INTEGER|4|tree_70441558298233.btree
sqlite> ALTER TABLE btt RENAME TO btt_renamed;
sqlite> .tables
btrees_mods_idxinfo  btt_renamed          btt_renamed_real
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt_renamed|2|0|id|INTEGER|4|tree_70441558298233.btree
sqlite> .exit

```

Рисунок 9. Переименование таблицы

Как мы видим, данное действие выполнено успешно. Таким образом, программа удовлетворяет данному функциональному требованию.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

6.3.8. Испытание выполнения требований к функциональным характеристикам в части алгоритма выбора индексирующей структуры данных и перестраивания дерева

Проведём испытание выполнения требований к функциональным характеристикам в части алгоритма выбора индексирующей структуры данных и перестраивания дерева. Создадим таблицу, как показано на скриншоте на рис. 10. Обратим внимание на второй столбец таблицы *btrees_mods_idxinfo* — в данном столбце хранится текущий тип используемого дерева (1 — В-дерево, 2 — В⁺-дерево, 3 — В*-дерево, 4 — В⁺*-дерево) — в данный момент используется В⁺-дерево.

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY);
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|2|0|id|INTEGER|4|tree_88211558298777.btree
```

Рисунок 10. Создание новой таблицы и проверка типа дерева

Вставим 1000 строк в таблицу. Как мы видим на скриншоте на рис. 11, тип дерева остался 2 (В⁺-дерево).

```
sqlite> INSERT INTO btt VALUES (989);
sqlite> INSERT INTO btt VALUES (990);
sqlite> INSERT INTO btt VALUES (991);
sqlite> INSERT INTO btt VALUES (992);
sqlite> INSERT INTO btt VALUES (993);
sqlite> INSERT INTO btt VALUES (994);
sqlite> INSERT INTO btt VALUES (995);
sqlite> INSERT INTO btt VALUES (996);
sqlite> INSERT INTO btt VALUES (997);
sqlite> INSERT INTO btt VALUES (998);
sqlite> INSERT INTO btt VALUES (999);
sqlite> INSERT INTO btt VALUES (1000);
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|2|0|id|INTEGER|4|tree_88211558298777.btree
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

Рисунок 11. Результат вставки 1000 строк в таблицу

Вставим 1001-ю строку в таблицу. Должно произойти перестраивание B^+ -дерева в B^* -дерево (3-й тип). Как показано на скриншоте на рис. 12, оно действительно выполнено.

```
sqlite> INSERT INTO btt VALUES (1001);
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|3|0|id|INTEGER|4|tree_88211558298777.btree
```

Рисунок 12. Результат вставки 1001-й строки в таблицу

При каждом удалении строки выполняется две операции с деревом — поиск удаляемого элемента и собственно его удаление. Вставим ещё одну строку в таблицу и произведём удаление первых (в порядке вставки) 499 строк из таблицы. Тип дерева не должен измениться, что действительно так, как показано на скриншоте на рис. 13.

```
sqlite> DELETE FROM btt WHERE id = 488;
sqlite> DELETE FROM btt WHERE id = 489;
sqlite> DELETE FROM btt WHERE id = 490;
sqlite> DELETE FROM btt WHERE id = 491;
sqlite> DELETE FROM btt WHERE id = 492;
sqlite> DELETE FROM btt WHERE id = 493;
sqlite> DELETE FROM btt WHERE id = 494;
sqlite> DELETE FROM btt WHERE id = 495;
sqlite> DELETE FROM btt WHERE id = 496;
sqlite> DELETE FROM btt WHERE id = 497;
sqlite> DELETE FROM btt WHERE id = 498;
sqlite> DELETE FROM btt WHERE id = 499;
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|3|0|id|INTEGER|4|tree_88211558298777.btree
```

Рисунок 13. Результат удаления первых (в порядке вставки) 499 строк из таблицы

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

Удалим 500-ю строку. При её удалении должно выполниться перестраивание В*-дерева в В⁺-дерево (4-й тип). Как видно на скриншоте на рис. 14, оно действительно выполнено.

```
sqlite> DELETE FROM btt WHERE id = 500;
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|4|0|id|INTEGER|4|tree_88211558298777.btree
```

Рисунок 14. Результат удаления 500-й строки из таблицы

Таким образом, программа удовлетворяет данному функциональному требованию.

6.3.9. Испытание выполнения требований к функциональным характеристикам в части сохранения таблицы, вместе с базой данных, на постоянном запоминающем устройстве, и открытия такой сохранённой таблицы

Проведём испытание выполнения требований к функциональным характеристикам в части сохранения таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вместе с базой данных на постоянном запоминающем устройстве, и открытия такой сохранённой таблицы.

Создадим таблицу и вставим в неё элементы, после чего сохраним её вместе с базой данных на постоянном запоминающем устройстве и выйдем из РСУБД SQLite, как показано на скриншоте на рис. 15.

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> INSERT INTO btt VALUES (4, 2, "ABC123"), (7, 1, "defwf");
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|ABC123
7|1|defwf
sqlite> .save db_47.sqlite
sqlite> .exit
```

Рисунок 15. Создание и сохранение таблицы вместе с базой данных на постоянном запоминающем устройстве

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

Вновь запустим PCУБД SQLite и попробуем открыть на диске сохранённую базу данных, а также проверим список таблиц содержимое таблицы *btt*, как показано на скриншоте на рис. 16.

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open db_47.sqlite
sqlite> .load ./btrees_mods
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|ABC123
7|1|defwf
sqlite>
```

Рисунок 16. Открытие сохранённой на диске базы данных и проверка содержимого

Как мы видим, сохранение на диске и открытие сохранённой таблицы вместе с базой данных успешно проведены, таким образом, программа удовлетворяет данным функциональным требованиям.

6.3.10. Испытание выполнения требований к функциональным характеристикам в части вывода графического изображения индексирующей структуры данных (дерева) таблицы в DOT-файл для GraphViz

Проведём испытание выполнения требований к функциональным характеристикам в части вывода графического изображения индексирующей структуры данных (дерева) таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, в DOT-файл для GraphViz.

Создадим таблицу, вставим в неё элементы, и применим к ней функцию *btreesModsVisualize*, как показано на скриншоте на рис. 17.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата


```

SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id TEXT PRIMARY KEY);
sqlite> INSERT INTO btt VALUES ("def456"), ("ABC123");
sqlite> SELECT btreesModsVisualize("btt", "btt_text.dot");
File written
sqlite> .exit

```

Рисунок 17. Использование функции *btreesModsVisualize* на таблице

В рабочем каталоге создан файл *btt_text.dot*. Преобразуем его в PNG-изображение командой `dot -Tpng btt_text.dot -o btt_text.png`. Получим изображение дерева, представленное на рис. 18.

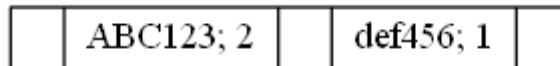


Рисунок 18. Изображение B⁺-дерева

Значение первичного ключа и row id соответствующей строки в таблицы с постфиксом *_real* разделены точкой с запятой. Более наглядный, но слишком крупный для помещения в настоящий документ, пример графического представления B-дерева, созданного при помощи данного расширения, можно найти на электронном носителе настоящей работы по пути *root\docs\images\btt_dot.png*.

Таким образом, программа удовлетворяет данному функциональному требованию.

6.3.11. Испытание выполнения требований к функциональным характеристикам в части вывода типа используемого в таблице дерева

Проведём испытание выполнения требований к функциональным характеристикам в части вывода типа дерева, используемого в таблице, использующей B-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.

Создадим таблицу и применим к ней функцию *btreesModsGetTreeType*, как показано на скриншоте на рис. 19.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY);
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> SELECT * FROM btrees_mods_idxinfo;
btt|2|0|id|INTEGER|4|tree_100291558299147.btree
sqlite> SELECT btreesModsGetTreeType("btt");
2

```

Рисунок 19. Использование функции *btreesModsGetTreeType* на таблице

Как мы видим, функция *btreesModsGetTreeType* вернула 2-й тип дерева, то есть B^+ -дерево, что, согласно данным в таблице *btrees_mods_idxinfo*, является истиной. Таким образом, программа удовлетворяет данному функциональному требованию.

6.3.12. Испытание выполнения требований к функциональным характеристикам в части вывода порядка используемого в таблице дерева

Проведём испытание выполнения требований к функциональным характеристикам в части вывода порядка дерева, используемого в таблице, использующей B -дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.

Создадим таблицу и применим к ней функцию *btreesModsGetTreeOrder*, как показано на скриншоте на рис. 20.

```

SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY);
sqlite> SELECT btreesModsGetTreeOrder("btt");
750

```

Рисунок 20. Использование функции *btreesModsGetTreeOrder* на таблице

Как мы видим, функция *btreesModsGetTreeOrder* вернула порядок дерева 750, что является истиной. Таким образом, программа удовлетворяет данному функциональному требованию.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

6.4. Испытание выполнения требований к надёжности

В ходе выполнения испытаний в пп. 6.2 — 6.3 настоящего документа было обеспечено стабильное и корректное функционирование компьютера и операционной системы. На протяжении всех испытаний в пп. 6.2 — 6.3 настоящего документа программа сохраняла работоспособность.

На скриншоте на рис. 21 видно, что программа обеспечивает проверку корректности входных данных и, при необходимости, выводит сообщение об ошибке, сохраняя при этом свою работоспособность.

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY);
sqlite> INSERT INTO btt VALUES ("ABC");
Error: datatype mismatch
sqlite>
```

Рисунок 21. Сообщение об ошибке при попытке вставить в таблицу данные некорректного типа

Таким образом, программа соответствует требованиям к надёжности.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 51 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО

Старший преподаватель департамента
программной инженерии факультета
компьютерных наук

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ С.А. Шершаков
«__» _____ 2019 г.

_____ В.В. Шилов
«__» _____ 2019 г.

КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ

Руководство оператора

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.02.09-01 34 01-1-ЛУ

Исполнитель
студент группы БПИ153
_____ / Ригин А. М. /
«__» _____ 2019 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.02.09-01 34 01-1-ЛУ

Москва 2019

УТВЕРЖДЕН
RU.17701729.02.09-01 34 01-1-ЛУ

**КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ**

Руководство оператора

RU.17701729.02.09-01 34 01-1

Листов 11

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.02.09-01 34 01-1				

Москва 2019

Содержание

1. Назначение и область применения	80
1.1. Назначение программы	80
1.1.1. Функциональное назначение	80
1.1.2. Эксплуатационное назначение	80
1.1. Состав функций программы	80
2. Условия выполнения программы.....	83
2.1. Требования к составу и параметрам технических средств	83
2.2. Требования к информационной и программной совместимости.....	83
2.3. Требования к квалификации пользователя	83
3. Выполнение программы	84
3.1. Запуск программы.....	84
3.2. Операции с виртуальной таблицей	84
3.3. Сохранение виртуальной таблицы вместе с базой данных на постоянном запоминающем устройстве и открытие сохранённой таким образом таблицы	85
3.4. Вывод данных об индексирующей структуре данных виртуальной таблицы и графического представления такой структуры данных	86
3.5. Сообщения программы об ошибках	87

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

1. Назначение и область применения

1.1. Назначение программы

«Компонент-расширение PCУБД SQLite для индексирования данных модификациями В-деревьев» может применяться для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, а также вывода графического представления используемого в качестве индексирующей структуры данных В-дерева или его модификации в DOT-файл для GraphViz и основных данных об используемом дереве.

1.1.1. Функциональное назначение

Программа может применяться для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, как расширение для PCУБД SQLite, позволяющее работать с таблицами, созданными при помощи данного расширения, и выводить графическое представление В-дерева или его модификации, используемой в данной таблице, в DOT-файл для GraphViz, и основные данные, связанные с соответствующей индексирующей структурой данных (деревом).

1.1.2. Эксплуатационное назначение

Программа будет применяться разработчиками и исследователями для индексирования данных модификациями В-деревьев (B^+ -дерево, B^* -дерево и B^{*+} -дерево) в реляционной СУБД SQLite, а также вывода графического представления используемого в качестве индексирующей структуры данных В-дерева или его модификации в DOT-файл для GraphViz и основных данных об используемом дереве, в том числе, в учебных и научных целях.

1.1. Состав функций программы

Расширение для SQLite (программа) обеспечивает возможность выполнения следующих функций:

1. Расширение позволяет создавать таблицу, использующую B^+ -дерево из данного расширения в качестве индексирующей структуры данных, с указанием столбца, являющегося первичным ключом таблицы.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2. Расширение позволяет удалять таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
3. Расширение позволяет производить поиск строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
4. Расширение позволяет производить вставку строки/строк в таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
5. Расширение позволяет производить удаление строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
6. Расширение позволяет производить обновление значений ячеек (включая ячейку с первичным ключом) строки/строк в таблице, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, по признаку равенства значения/значений первичного ключа искомой/искомых строки/строк таблицы заданному значению/заданным значениям.
7. Расширение позволяет переименовывать таблицу, использующую В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
8. Расширение при каждой операции с таблицей, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, запускает алгоритм выбора индексирующей структуры данных из модификаций В-дерева (B^+ -дерева, B^* -дерева и B^{*+} -дерева) и перестраивает имеющуюся индексирующую структуру данных на новую (если была выбрана новая), сохраняя все имеющиеся в ней данные.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

9. Расширение поддерживает сохранение таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вместе с базой данных на постоянном запоминающем устройстве.
10. Расширение поддерживает открытие сохранённой вместе с базой данных на постоянном запоминающем устройстве таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных.
11. Расширение поддерживает для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод графического представления индексирующей структуры данных (дерева) таблицы в DOT-файл для GraphViz.
12. Расширение поддерживает для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод типа используемого дерева (1 — В-дерево, 2 — В⁺-дерево, 3 — В^{*}-дерево, 4 — В⁺⁺-дерево).
13. Расширение поддерживает для таблицы, использующей В-дерево или его модификацию из данного расширения в качестве индексирующей структуры данных, вывод порядка используемого дерева.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2. Условия выполнения программы

2.1. Требования к составу и параметрам технических средств

Для нормального функционирования программы требуется компьютер, оснащенный следующими техническими компонентами:

- 1) процессор не ниже Intel Pentium/Celeron, AMD K6/Athlon/Duron или совместимый с ними с тактовой частотой не ниже 1 ГГц;
- 2) 512 Мб ОЗУ или более;
- 3) жесткий диск с объемом свободной памяти не менее 100 Мб;
- 4) VGA-совместимые видеоадаптер и монитор;
- 5) клавиатура.

2.2. Требования к информационной и программной совместимости

Для нормального функционирования программы требуется компьютер, оснащенный следующими программными компонентами:

- 1) операционная система Microsoft Windows 7 / 8 / 8.1 / 10 либо Ubuntu версии 16.04 LTS или выше;
- 2) SQLite 3 версии 3.24 или выше.

2.3. Требования к квалификации пользователя

Требуемая квалификация пользователя программы — оператор ЭВМ с базовыми знаниями в области работы с системами управления базами данных (СУБД).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

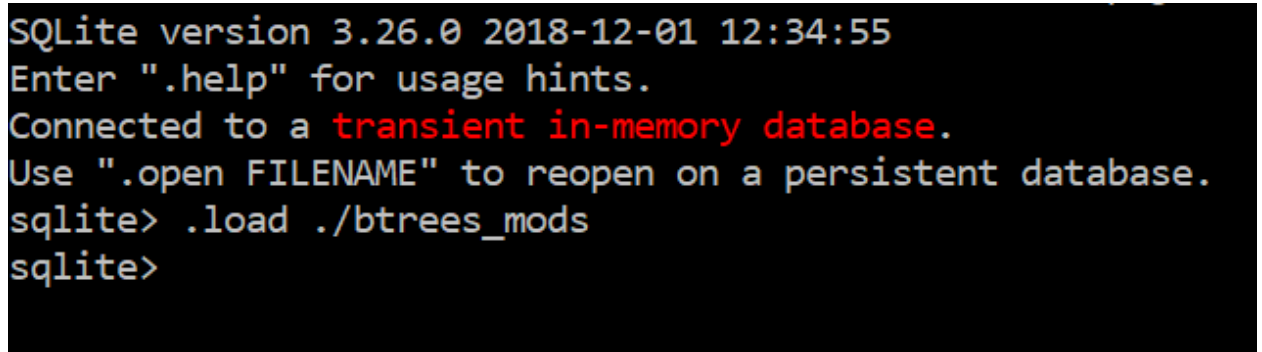
3. Выполнение программы

3.1. Запуск программы

Программа поставляется пользователю на электронном носителе информации в виде динамической библиотеки — файл с расширением `.dll` (для Windows) либо `.so` (для Linux).

Программа сразу готова к запуску, её установка не требуется.

Чтобы запустить программу, необходимо запустить РСУБД SQLite и загрузить расширение командой `.load` с указанием пути к файлу расширения. Например, команда может выглядеть как `.load ~/btrees_mods.dll`. Если файл расширения находится в текущем рабочем каталоге и называется `btrees_mods.dll` (для Windows) либо `btrees_mods.so` (для Linux), то команда будет выглядеть как `.load ./btrees_mods`. Данное действие показано на скриншоте на рис. 1.



```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite>
```

Рисунок 1. Загрузка расширения в SQLite

3.2. Операции с виртуальной таблицей

Для создания таблицы, использующей расширение `btrees_mods` (далее — виртуальная таблица), необходимо выполнить команду вида `CREATE VIRTUAL TABLE tableName USING btrees_mods (args);`, где `tableName` — имя создаваемой виртуальной таблицы, `args` — аргументы таблицы (имена и типы столбцов и т. д.). В аргументах надо указать столбец, являющийся первичным ключом виртуальной таблицы. Например, `CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY, a INTEGER, b TEXT);`.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

С виртуальной таблицей, как и с любой другой таблицей РСУБД SQLite, можно выполнять различные действия, используя запросы языка SQL. Поддерживаются: вставка строки/строк в таблицу (*INSERT INTO ...*), удаление строки/строк из таблицы (*DELETE FROM ...*), обновление значений ячейки/ячеек строки/строк в таблице (*UPDATE ...*), поиск строки/строк в таблице (*SELECT ...*) — только по признаку равенства значения первичного ключа таблицы заданному значению, удаление таблицы (*DROP TABLE ...*) и переименование таблицы (*ALTER TABLE ... RENAME TO ...*). Примеры этих действий, а также пример создания виртуальной таблицы, показаны на скриншоте на рис. 2.

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY, a INTEGER, b TEXT);
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
sqlite> INSERT INTO btt VALUES (4, 2, "AB42"), (7, 1, "DE47");
sqlite> SELECT * FROM btt WHERE id = 4;
4|2|AB42
sqlite> SELECT * FROM btt WHERE id = 7;
7|1|DE47
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|AB42
7|1|DE47
sqlite> UPDATE btt SET b = "new text" WHERE id = 4;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|new text
7|1|DE47
sqlite> DELETE FROM btt WHERE id = 7;
sqlite> SELECT * FROM btt WHERE id = 4 OR id = 7;
4|2|new text
sqlite> ALTER TABLE btt RENAME TO btt_renamed;
sqlite> .tables
btrees_mods_idxinfo  btt_renamed                btt_renamed_real
sqlite> DROP TABLE btt_renamed;
sqlite> .tables
btrees_mods_idxinfo
sqlite> .exit
```

Рисунок 2. Действия с виртуальной таблицей

3.3. Сохранение виртуальной таблицы вместе с базой данных на постоянном запоминающем устройстве и открытие сохранённой таким образом таблицы

Для того, чтобы сохранить виртуальную таблицу вместе с базой данных на постоянном запоминающем устройстве (например, жёстком диске), необходимо ввести

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

команду `.save <имя_сохраняемого_файла>` . После этого можно выйти из PCУБД SQLite.

Пример этих действий представлен на скриншоте на рис. 3.

```
sqlite> .save db_btt_test.sqlite
sqlite> .exit
```

Рисунок 3. Сохранение виртуальной таблицы вместе с базой данных на постоянном запоминающем устройстве

Таблицу, сохранённую таким образом, позже можно открыть вместе с базой данных, в которой она была сохранена. Для этого необходимо ввести команду `.open <имя_сохранённого_файла>` . После этого необходимо повторно загрузить расширение при помощи действий, описанных в п. 3.1 настоящего документа. Пример этих действий приведён на скриншоте на рис. 4.

```
sqlite> .open db_btt_test.sqlite
sqlite> .load ./btrees_mods
sqlite> .tables
btrees_mods_idxinfo  btt                                btt_real
```

Рисунок 4. Открытие сохранённой вместе с базой данных на постоянном запоминающем устройстве виртуальной таблицы

3.4. Вывод данных об индексирующей структуре данных виртуальной таблицы и графического представления такой структуры данных

Для записи графического представления индексирующей структуры данных (дерева) виртуальной таблицы в DOT-файл для GraphViz необходимо воспользоваться функцией `btreesModsVisualize` и выполнить SQL-запрос вида `SELECT btreesModsVisualize("<имя_вирт_таблицы>", "<имя_сохраняемого_DOT_файла>");` . Например, `SELECT btreesModsVisualize("btt", "btt_dot.dot");` . В случае успешного выполнения запроса выведется надпись «File written», в противном случае, выведется информация об ошибке. Пример выполнения данного запроса представлен на скриншоте на рис. 5.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```
sqlite> SELECT btreesModsVisualize("btt", "btt_dot.dot");
File written
```

Рисунок 5. Использование функции *btreesModsVisualize* на виртуальной таблице

Далее полученный DOT-файл можно, при необходимости, сконвертировать в изображение нужного формата при помощи CLI-команд GraphViz. Например, *dot -Tpng btt_dot.dot -o btt_dot.png*.

Для вывода типа (1 — B-дерево, 2 — B⁺-дерево, 3 — B^{*}-дерево, 4 — B⁺⁺-дерево) используемого в качестве индексирующей структуры данных сильно ветвящегося дерева необходимо воспользоваться функцией *btreesModsGetTreeType* и выполнить SQL-запрос вида *SELECT btreesModsGetTreeType("<имя_вирт_таблицы>")*. Например, *SELECT btreesModsGetTreeType("btt");*. Пример выполнения данного запроса приведён на скриншоте на рис. 6. В данном случае используется B⁺-дерево.

```
sqlite> SELECT btreesModsGetTreeType("btt");
2
```

Рисунок 6. Использование функции *btreesModsGetTreeType* на виртуальной таблице

Для вывода порядка используемого в качестве индексирующей структуры данных сильно ветвящегося дерева необходимо воспользоваться функцией *btreesModsGetTreeOrder* и выполнить SQL-запрос вида *SELECT btreesModsGetTreeOrder("<имя_вирт_таблицы>")*. Например, *SELECT btreesModsGetTreeOrder("btt");*. Пример выполнения данного запроса приведён на скриншоте на рис. 7.

```
sqlite> SELECT btreesModsGetTreeOrder("btt");
750
```

Рисунок 7. Использование функции *btreesModsGetTreeOrder* на виртуальной таблице

3.5. Сообщения программы об ошибках

В случае ввода некорректных входных данных программа сообщает об ошибке. Это может выглядеть, например, как на скриншоте на рис. 8. В данном случае произведена попытка вставить в виртуальную таблицу данные некорректного типа.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .load ./btrees_mods
sqlite> CREATE VIRTUAL TABLE btt USING btrees_mods (id INTEGER PRIMARY KEY);
sqlite> INSERT INTO btt VALUES ("ABC");
Error: datatype mismatch
sqlite>

```

Рисунок 8. Сообщение об ошибке

При появлении таких ошибок необходимо попытаться устранить ошибку и выполнить действие, которое привело к ошибке, заново.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 34 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО

Старший преподаватель департамента
программной инженерии факультета
компьютерных наук

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

_____ С.А. Шершаков
«__» _____ 2019 г.

_____ В.В. Шилов
«__» _____ 2019 г.

**КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ**

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.02.09-01 12 01-1-ЛУ

Исполнитель
студент группы БПИ153
_____ / Ригин А. М. /
«__» _____ 2019 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.02.09-01 12 01-1-ЛУ

Москва 2019

УТВЕРЖДЕН
RU.17701729.02.09-01 12 01-1-ЛУ

**КОМПОНЕНТ-РАСШИРЕНИЕ РСУБД SQLITE ДЛЯ
ИНДЕКСИРОВАНИЯ ДАННЫХ МОДИФИКАЦИЯМИ В-ДЕРЕВЬЕВ**

Текст программы

RU.17701729.02.09-01 12 01-1

Листов 47

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.02.09-01 12 01-1				

Москва 2019

Содержание

1. Файл btree_c.h	92
2. Файл btree_c.hpp	98
3. Файл btrees_mods.h	101
4. Файл btrees_mods.c	114

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

1. Файл btree_c.h

Данный файл является заголовочным файлом для API на C для имеющейся C++-библиотеки сильно ветвящихся деревьев.

```

/// \file
/// \brief      B-tree and modifications C-API.
/// \authors    Anton Rigin
/// \version    0.1.0
/// \date       22.12.2018 -- 04.05.2019
///            The bachelor thesis of Anton Rigin,
///            the HSE Software Engineering 4-th year bachelor student.
///
////////////////////////////////////
///

#ifndef BTREES_BTREE_C_H
#define BTREES_BTREE_C_H

#include "btree.h" // The B-tree modifications C++ library.

using namespace btree;

#define INTEGER_SIZE 4
#define FLOAT_SIZE 8
#define TEXT_SIZE 256
#define NULL_SIZE 1

/**
 * The simple byte comparator for the tree.
 */
struct ByteComparator : public BaseBTree::IComparator {

    /**
     * The bytes count of the first part (the primary key value part) of the
     tree keys.
     */
    UInt firstPartBytes = 0;

    virtual bool compare(const Byte* lhv, const Byte* rhv, UInt sz) override;

    virtual bool isEqual(const Byte* lhv, const Byte* rhv, UInt sz) override;
}; // struct ByteComparator

typedef struct ByteComparator ByteComparator;

/**
 * The byte comparator for the tree which searches for all the tree keys.
 */
struct SearchAllByteComparator : public BaseBTree::IComparator {

    /**
     * The bytes count of the first part (the primary key value part) of the
     tree keys.
     */
    UInt firstPartBytes = 0;

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

virtual bool compare(const Byte* lhv, const Byte* rhv, UInt sz) override;

virtual bool isEqual(const Byte* lhv, const Byte* rhv, UInt sz) override
{ return true; }
}; // struct SearchAllByteComparator

typedef struct SearchAllByteComparator SearchAllByteComparator;

/**
 * The byte printer for writing the key into the GraphViz file byte-by-byte.
 */
struct BytePrinter : public BaseBTree::IKeyPrinter {

    UInt firstPartBytes = TEXT_SIZE;

    virtual std::string print(const Byte* key, UInt sz) override;
}; // struct BytePrinter

typedef struct BytePrinter BytePrinter;

/**
 * The integer byte printer for writing the integer key into the GraphViz
 file.
 */
struct IntBytePrinter : public BaseBTree::IKeyPrinter {

    UInt firstPartBytes = INTEGER_SIZE;

    virtual std::string print(const Byte* key, UInt sz) override
    { return std::to_string(*((int*) key)) + std::string("; ") +
      std::to_string(*((long long*) &key[firstPartBytes])); }
}; // struct IntBytePrinter

typedef struct IntBytePrinter IntBytePrinter;

/**
 * The float byte printer for writing the float key into the GraphViz file.
 */
struct FloatBytePrinter : public BaseBTree::IKeyPrinter {

    UInt firstPartBytes = FLOAT_SIZE;

    virtual std::string print(const Byte* key, UInt sz) override
    { return std::to_string(*((double*) key)) + std::string("; ") +
      std::to_string(*((long long*) &key[firstPartBytes])); }
}; // struct FloatBytePrinter

typedef struct FloatBytePrinter FloatBytePrinter;

/**
 * The null byte printer for writing the null key into the GraphViz file.
 */
struct NullBytePrinter : public BaseBTree::IKeyPrinter {

    UInt firstPartBytes = NULL_SIZE;

    virtual std::string print(const Byte* key, UInt sz) override { return
std::string("NULL"); }
}; // struct NullBytePrinter

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

typedef struct NullBytePrinter NullBytePrinter;

#ifdef __cplusplus
extern "C" {
#endif

static ByteComparator byteComparator;

static SearchAllByteComparator searchAllByteComparator;

static BytePrinter bytePrinter;

static IntBytePrinter intBytePrinter;

static FloatBytePrinter floatBytePrinter;

static NullBytePrinter nullBytePrinter;

/**
 * Creates the tree of the given type (B-tree or one of its modifications).
 *
 * @param pTree The pointer for saving the created tree.
 * @param treeType The type of the created tree (B-tree or one of its
modifications).
 * @param order The tree order.
 * @param keySize The tree key size.
 * @param treeFileName The tree file name.
 */
static void create(FileBaseBTree** pTree, BaseBTree::TreeType treeType,
    UShort order, UShort keySize, const char* treeFileName);

/**
 * Creates the B-tree.
 *
 * @param pTree The pointer for saving the created tree.
 * @param order The tree order.
 * @param keySize The tree key size.
 * @param treeFileName The tree file name.
 */
static void createBTree(FileBaseBTree** pTree, UShort order, UShort keySize,
const char* treeFileName);

/**
 * Open the tree of the given type (B-tree or one of its modifications).
 *
 * @param pTree The pointer for saving the opened tree.
 * @param treeType The type of the opened tree (B-tree or one of its
modifications).
 * @param treeFileName The tree file name.
 */
static void open(FileBaseBTree** pTree, BaseBTree::TreeType treeType, const
char* treeFileName);

/**
 * Closes the tree.
 *
 * @param pTree The pointer of tree being closed.
 */

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

static void close(FileBaseBTree** pTree);

/**
 * Sets the simple byte comparator to the given tree.
 *
 * @param tree The pointer to the given tree.
 */
static void setByteComparator(FileBaseBTree* tree) { tree->getTree()-
>setComparator(&byteComparator); }

/**
 * Sets the byte comparator which searches for all the tree keys to the given
 * tree.
 *
 * @param tree The pointer to the given tree.
 */
static void setSearchAllByteComparator(FileBaseBTree* tree)
    { tree->getTree()->setComparator(&searchAllByteComparator); }

/**
 * Inserts the key into the tree.
 *
 * @param tree The pointer to the tree.
 * @param k The inserted key.
 */
static void insert(FileBaseBTree* tree, const Byte* k) { tree->insert(k); }

/**
 * Searches for the key in the tree.
 *
 * @param tree The pointer to the tree.
 * @param k The searched key.
 * @return The found key.
 */
static Byte* search(FileBaseBTree* tree, const Byte* k) { return tree-
>search(k); }

/**
 * Searches for all the given keys in the tree.
 *
 * @param tree The pointer to the tree.
 * @param k The searched key.
 * @param keysPointer The pointers to the found keys array.
 * @return The found keys count.
 */
static int searchAll(FileBaseBTree* tree, const Byte* k, Byte***
keysPointer);

#ifdef BTREE_WITH_DELETION

/**
 * Removes the key from the tree.
 *
 * @param tree The pointer to the tree.
 * @param k The removed key.
 * @return true if the key was successfully removed, false otherwise.
 */
static bool removeKey(FileBaseBTree* tree, const Byte* k) { return tree-
>remove(k); }

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

/**
 * Removes all the given keys from the tree.
 *
 * @param tree The pointer to the tree.
 * @param k The removed key.
 * @return The removed keys count.
 */
static int removeAll(FileBaseBTree* tree, const Byte* k) { return tree->removeAll(k); }

#endif

/**
 * Visualizes the tree to the GraphViz DOT file.
 *
 * @param tree The pointer to the tree.
 * @param dotFileName The GraphViz DOT file name.
 * @return true if the DOT file is written, false if it is impossible to open
the DOT file for writing.
 */
static bool visualize(FileBaseBTree* tree, const char* dotFileName);

/**
 * Gets order for the tree.
 *
 * @param tree The pointer to the tree.
 * @return The tree order.
 */
static int getOrder(FileBaseBTree* tree) { return tree->getTree()->getOrder(); }

/**
 * Sets the byte printer as the key printer to the tree.
 *
 * @param tree The pointer to the tree.
 */
static void setBytePrinter(FileBaseBTree* tree) { tree->getTree()->setKeyPrinter(&bytePrinter); }

/**
 * Sets the integer byte printer as the key printer to the tree.
 *
 * @param tree The pointer to the tree.
 */
static void setIntBytePrinter(FileBaseBTree* tree) { tree->getTree()->setKeyPrinter(&intBytePrinter); }

/**
 * Sets the float byte printer as the key printer to the tree.
 *
 * @param tree The pointer to the tree.
 */
static void setFloatBytePrinter(FileBaseBTree* tree) { tree->getTree()->setKeyPrinter(&floatBytePrinter); }

/**
 * Sets the null printer as the key printer to the tree.
 *

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

* @param tree The pointer to the tree.
*/
static void setNullBytePrinter(FileBaseBTree* tree) { tree->getTree()-
>setKeyPrinter(&nullBytePrinter); }

#ifdef __cplusplus
}; // extern "C"
#endif

#include "btree_c.hpp"

#endif //BTREES_BTREE_C_H

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

2. Файл btree_c.hpp

Данный файл является файлом с реализацией методов, описанных в файле btree_c.h — заголовочном файле для API на C для имеющейся C++-библиотеки сильно ветвящихся деревьев.

```

/// \file
/// \brief      B-tree and modifications C-API.
/// \authors    Anton Rigin
/// \version    0.1.0
/// \date       22.12.2018 -- 04.05.2019
///
///            The bachelor thesis of Anton Rigin,
///            the HSE Software Engineering 4-th year bachelor student.
///
////////////////////////////////////
///

#include "btree_c.h"

bool ByteComparator::compare(const Byte *lhv, const Byte *rhv, UInt sz)
{
    for (UInt i = 0; i < sz && i < firstPartBytes; ++i)
    {
        if (lhv[i] < rhv[i])
            return true;
        if (lhv[i] > rhv[i])
            return false;
    }

    return false;
}

bool ByteComparator::isEqual(const Byte *lhv, const Byte *rhv, UInt sz)
{
    for (UInt i = 0; i < sz && i < firstPartBytes; ++i)
        if (lhv[i] != rhv[i])
            return false;

    return true;
}

bool SearchAllByteComparator::compare(const Byte *lhv, const Byte *rhv, UInt
sz)
{
    for (UInt i = 0; i < sz && i < firstPartBytes; ++i)
    {
        if (lhv[i] < rhv[i])
            return true;
        if (lhv[i] > rhv[i])
            return false;
    }

    return false;
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

std::string BytePrinter::print(const Byte *key, UInt sz)
{
    std::string result;

    for (UInt i = 0; i < sz && i < firstPartBytes && key[i] != 0; ++i)
        result += std::string(1, *((char*) &key[i]));

    result += std::string("; ") + std::to_string(*((long long*)
&key[firstPartBytes]));

    return result;
}

#ifdef __cplusplus
extern "C" {
#endif

static void create(FileBaseBTree** pTree, BaseBTree::TreeType treeType,
    UShort order, UShort keySize, const char* treeFileName)
{
    close(pTree);

    try
    {
        *pTree = new FileBaseBTree(treeType, order, keySize, &byteComparator,
treeFileName);
    }
    catch (std::runtime_error&)
    {
        *pTree = nullptr;
    }
}

static void createBTree(FileBaseBTree** pTree, UShort order, UShort keySize,
const char* treeFileName)
{
    create(pTree, BaseBTree::TreeType::B_TREE, order, keySize, treeFileName);
}

static void open(FileBaseBTree** pTree, BaseBTree::TreeType treeType, const
char* treeFileName)
{
    close(pTree);

    try
    {
        *pTree = new FileBaseBTree(treeType, treeFileName, &byteComparator);
    }
    catch (std::runtime_error&)
    {
        *pTree = nullptr;
    }
}

static void close(FileBaseBTree** pTree)
{
    if (*pTree != nullptr)
    {

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

        delete *pTree;
        *pTree = nullptr;
    }
}

static int searchAll(FileBaseBTree* tree, const Byte* k, Byte*** keysPointer)
{
    std::list<Byte*> keys;
    int result = tree->searchAll(k, keys);

    *keysPointer = (Byte**) malloc(sizeof(Byte*) * keys.size());

    std::list<Byte*>::iterator iter = keys.begin();
    for (int i = 0; i < keys.size() && iter != keys.end(); ++i, ++iter)
        (*keysPointer)[i] = *iter;

    return result;
}

static bool visualize(FileBaseBTree* tree, const char* dotFileName)
{
    std::ofstream dotFile(dotFileName);

    if (!dotFile.is_open())
        return false;

    tree->getTree()->writeDot(dotFile);

    dotFile.close();

    return true;
}

#ifdef __cplusplus
} // extern "C"
#endif

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

3. Файл btrees_mods.h

Данный файл является заголовочным файлом для расширения для SQLite.

```

/// \file
/// \brief      B-tree and modifications SQLite extension.
/// \authors    Anton Rigin
/// \version    0.1.0
/// \date       03.01.2019 -- 04.05.2019
///            The bachelor thesis of Anton Rigin,
///            the HSE Software Engineering 4-th year bachelor student.
///
/////////////////////////////////////////////////////////////////
///

#ifndef BTREES_BTREES_MODS_H
#define BTREES_BTREES_MODS_H

#include "stdlib.h"
#include "stdio.h"
#include "time.h"
#include "string.h"
#include "sqlite3ext.h"
#include "btree_c.h"

SQLITE_EXTENSION_INIT1

using namespace btree;

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _WIN32
__declspec(dllexport)
#endif

int sqlite3_btreesmods_init(sqlite3* db, char** pErrMsg, const
sqlite3_api_routines* pApi);

#ifdef __cplusplus
}; // extern "C"
#endif

#define CHAR_BUFFER_SIZE 256

#define BTREE_NUM 1
#define BPLUSTREE_NUM 2
#define BSTARTREE_NUM 3
#define BSTARPLUSTREE_NUM 4

#define INTEGER_SIZE 4
#define FLOAT_SIZE 8
#define TEXT_SIZE 256
#define BLOB_SIZE 256

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

#define NULL_SIZE 1

#define ROWID_SIZE 8

#define ERROR_CODE -1

#define ROWID_IDX_EOF -1

#define TRUE 1
#define FALSE 0

#define TREE_ORDER 750

#define REBUILD_COEF 0.1
#define REBUILD_SPLIT_PERCENT_POINT 0.7397
#define REBUILD_COUNT 1000
#define REBUILD_MAX_COUNT 10000

/**
 * The virtual table's index params.
 */
struct indexParams {

    /**
     * The best B-tree modification based index structure number.
     */
    int bestIndex;

    /**
     * The number of the index column (the primary key column).
     */
    int indexColNumber;

    /**
     * The name of the index column (the primary key column).
     */
    char* indexColName = NULL;

    /**
     * The data type name of the index column (the primary key column).
     */
    char* indexDataType = NULL;

    /**
     * The data size of the index column (the primary key column).
     */
    int indexDataSize;

    /**
     * The index tree's file name.
     */
    char* treeFileName = NULL;
};

typedef struct indexParams indexParams;

/**
 * The index usage statistics.
 */

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

struct indexStats {

    /**
     * The index searches count.
     */
    int searchesCount;

    /**
     * The index inserts count.
     */
    int insertsCount;

    /**
     * The index deletes count.
     */
    int deletesCount;

    /**
     * 1 if the statistics was created when the virtual table was created, 0
    otherwise.
     */
    int isOriginalStats;
};

typedef struct indexStats indexStats;

/**
 * The btrees_mods module's virtual table.
 */
struct btreesModsVirtualTable {

    /**
     * The base class. Must be first.
     */
    sqlite3_vtab base;

    /**
     * The SQLite DB connection.
     */
    sqlite3* db = NULL;

    /**
     * The virtual table name.
     */
    char* tableName = NULL;

    /**
     * The virtual table index tree.
     */
    FileBaseBTree* tree = NULL;

    /**
     * The virtual table index params.
     */
    indexParams params = {
        BPLUSTREE_NUM,
        -1,
        NULL,
        NULL,
    }
};

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

        0,
        NULL
    };

    /**
     * The virtual table index statistics.
     */
    indexStats stats = {
        0,
        0,
        0,
        FALSE
    };
};

typedef struct btreesModsVirtualTable btreesModsVirtualTable;

/**
 * The btrees_mods module cursor containing the search results.
 */
struct btreesModsCursor {

    /**
     * The base class. Must be first.
     */
    sqlite3_vtab_cursor base;

    /**
     * The ids of the found rows.
     */
    sqlite_int64* rowsIds = NULL;

    /**
     * The current row id index in the rowsIds array.
     */
    int currentRowIdIdx;

    /**
     * The rowsIds array size.
     */
    int rowsIdsCount;
};

typedef struct btreesModsCursor btreesModsCursor;

/**
 * Creates the virtual table (after the "CREATE TABLE" query).
 *
 * @param db The SQLite DB connection.
 * @param pAux The copy of the client data pointer that was the fourth
 * argument to the sqlite3_create_module() call
 * that registered the virtual table module.
 * @param argc The count of the arguments for the virtual table creating.
 * @param argv The arguments for the virtual table creating.
 * @param ppVTab The pointer for saving the virtual table.
 * @param pzErr The pointer for writing the error message.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsCreate(sqlite3* db, void* pAux, int argc, const char*

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

const* argv,
    sqlite3_vtab** ppVTab, char** pzErr);

/**
 * Connects to the existing virtual table.
 *
 * @param db The SQLite DB connection.
 * @param pAux The copy of the client data pointer that was the fourth
argument to the sqlite3_create_module() call
 * that registered the virtual table module.
 * @param argc The count of the arguments for the virtual table creating.
 * @param argv The arguments for the virtual table creating.
 * @param ppVTab The pointer for saving the virtual table.
 * @param pzErr The pointer for writing the error message.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsConnect(sqlite3* db, void* pAux, int argc, const char*
const* argv,
    sqlite3_vtab** ppVTab, char** pzErr);

/**
 * Initializes the virtual table.
 *
 * @param db The SQLite DB connection.
 * @param pAux The copy of the client data pointer that was the fourth
argument to the sqlite3_create_module() call
 * that registered the virtual table module.
 * @param argc The count of the arguments for the virtual table creating.
 * @param argv The arguments for the virtual table creating.
 * @param ppVTab The pointer for saving the virtual table.
 * @param pzErr The pointer for writing the error message.
 * @param isCreate 1 is the virtual tables is creating, 0 otherwise.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsInit(sqlite3* db, void* pAux, int argc, const char*
const* argv,
    sqlite3_vtab** ppVTab, char** pzErr, int isCreate);

/**
 * Prepares the virtual table for searching.
 *
 * @param tab The virtual table instance.
 * @param pIdxInfo The virtual table index info instance.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsBestIndex(sqlite3_vtab* tab, sqlite3_index_info*
pIdxInfo);

/**
 * Disconnects from the virtual table.
 *
 * @param pVtab The virtual table instance.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsDisconnect(sqlite3_vtab* pVtab);

/**
 * Destroys the virtual table (after the "DROP TABLE" query).
 *

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата


```

* @param pVtab The virtual table instance.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsDestroy(sqlite3_vtab* pVtab);

/**
* Searches for rows in the table.
*
* @param cursor The SQLite virtual table cursor instance.
* @param idxNum The best index structure number.
* @param idxStr The idxStr string prepared by btreesModsBestIndex();
* @param argc The count of the constraint values for searching.
* @param argv The constraint values for searching.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsFilter(sqlite3_vtab_cursor* cursor, int idxNum, const
char* idxStr,
    int argc, sqlite3_value** argv);

/**
* Searches for the next row matching the search constraints.
*
* @param cursor The SQLite virtual table cursor instance.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsNext(sqlite3_vtab_cursor* cursor);

/**
* Returns 1 if no more rows match the search constraints, 0 otherwise.
*
* @param cursor The SQLite virtual table cursor instance.
* @return 1 if no more rows match the search constraints, 0 otherwise.
*/
static int btreesModsEof(sqlite3_vtab_cursor* cursor);

/**
* Extracts the @param n-th column from the found row.
*
* @param cursor The SQLite virtual table cursor instance.
* @param context The SQLite context instance for saving the cell value.
* @param n The number of column to be extracted from the row.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsColumn(sqlite3_vtab_cursor* cursor, sqlite3_context*
context, int n);

/**
* Handles the search constraint.
*
* @param cursor The btrees_mods module cursor instance.
* @param columnNum The constraint's column number.
* @param operation The constraint's operation.
* @param exprValue The constraint's right side expression value.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsHandleConstraint(btreesModsCursor* cursor, int
columnNum, unsigned char operation,
    sqlite3_value* exprValue);

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

/**
 * Handles the equality search constraint.
 *
 * @param cursor The btrees_mods module cursor instance.
 * @param exprValue The constraint's right side expression value.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsHandleConstraintEq(btreesModsCursor* cursor,
sqlite3_value* exprValue);

/**
 * Updates the row in the virtual table (inserts, updates or deletes the
row).
 *
 * @param pVTab The virtual table instance.
 * @param argc The count of the arguments for updating.
 * @param argv The arguments for updating.
 * @param pRowid The pointer for saving the row id of the updated row.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsUpdate(sqlite3_vtab* pVTab, int argc, sqlite3_value**
argv, sqlite_int64* pRowid);

/**
 * Updates the row in the virtual table (after the "UPDATE" query).
 *
 * @param pVTab The virtual table instance.
 * @param argc The count of the arguments for updating.
 * @param argv The arguments for updating.
 * @param pRowid The pointer for saving the row id of the updated row.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsDoUpdate(sqlite3_vtab* pVTab, int argc, sqlite3_value**
argv, sqlite_int64* pRowid);

/**
 * Deletes the row from the virtual table (after the "DELETE FROM" query).
 *
 * @param pVTab The virtual table instance.
 * @param primaryKeyValue The primary key value of the deleted row.
 * @param pRowid The pointer for saving the row id of the deleted row.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsDelete(sqlite3_vtab* pVTab, sqlite3_value*
primaryKeyValue, sqlite_int64* pRowid);

/**
 * Inserts the row into the virtual table (after the "INSERT INTO" query).
 *
 * @param pVTab The virtual table instance.
 * @param argc The count of the arguments for inserting.
 * @param argv The arguments for inserting.
 * @param pRowid The pointer for saving the row id of the inserted row.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int btreesModsInsert(sqlite3_vtab* pVTab, int argc, sqlite3_value**
argv, sqlite_int64* pRowid);

/**

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

* Opens the cursor for the searching in the virtual table.
*
* @param pVTab The virtual table instance.
* @param ppCursor The pointer for saving the opened cursor.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsOpen(sqlite3_vtab* pVTab, sqlite3_vtab_cursor**
ppCursor);

/**
* The closes the cursor after the searching in the virtual table.
*
* @param pCur The cursor for closing.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsClose(sqlite3_vtab_cursor* pCur);

/**
* Determines the row id of the row pointed by the cursor.
*
* @param pCur The cursor instance.
* @param pRowid The pointer for saving the determined row id.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsRowid(sqlite3_vtab_cursor *pCur, sqlite_int64 *pRowid);

/**
* Renames the virtual table.
*
* @param pVtab The virtual table instance.
* @param zNew The new name for the virtual table.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int btreesModsRename(sqlite3_vtab* pVtab, const char* zNew);

/**
* Generates the tree file name.
*
* @param treeFileName The pointer for saving the generated tree file name.
* @return The generated tree file name.
*/
static char* getTreeFileName(char* treeFileName);

/**
* Creates the B-tree modification based index for the virtual table.
*
* @param virtualTable The virtual table instance.
* @param order The tree order.
* @param keySize The tree key size.
* @return SQLITE_OK if successful, SQLite error code otherwise.
*/
static int createIndex(btreesModsVirtualTable* virtualTable, int order, int
keySize);

/**
* Creates the virtual table's B-tree modification based index.
*
* @param virtualTable The virtual table instance.
* @return SQLITE_OK if successful, SQLite error code otherwise.

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

*/
static int openIndex(btreesModsVirtualTable* virtualTable);

/**
 * Registers the virtual table's index column (the primary key column).
 *
 * @param db The SQLite DB connection.
 * @param stmt The SQLite SELECT statement performed on the real table
representation of the virtual table.
 * @param virtualTable The virtual table instance.
 * @param tableName The virtual table name.
 * @param treeFileName The index tree file name.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int registerIndexColumn(sqlite3* db, sqlite3_stmt* stmt,
btreesModsVirtualTable* virtualTable,
    const char* tableName, const char* treeFileName);

/**
 * Converts the data type name to the data size.
 *
 * @param dataType The data type name.
 * @return The data size.
 */
static int getDataSizeByType(const char* dataType);

/**
 * Converts the data type number to the data type name.
 *
 * @param dataType The data type number.
 * @return The data type name.
 */
static const char* getDataTypeByInt(int dataType);

/**
 * Converts the data type name to the data type number.
 *
 * @param dataType The data type name.
 * @return The data type number.
 */
static int getIntByDataType(const char* dataType);

/**
 * Gets the row id for the given primary key value.
 *
 * @param virtualTable The virtual table instance.
 * @param primaryKeyValue The primary key value.
 * @return The row id.
 */
static sqlite3_int64 getRowId(btreesModsVirtualTable* virtualTable,
sqlite3_value* primaryKeyValue);

/**
 * Executes the SQL query and finalizes its SQLite statement.
 *
 * @param db The SQLite DB connection.
 * @param sql The SQL query.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

static int executeSqlAndFinalize(sqlite3* db, char* sql);

/**
 * Executes the SQL query.
 *
 * @param db The SQLite DB connection.
 * @param sql The SQL query.
 * @param stmt The pointer for saving the query's SQLite statement.
 * @return SQLITE_OK if successful, SQLite error code otherwise.
 */
static int executeSql(sqlite3* db, char* sql, sqlite3_stmt** stmt);

/**
 * Creates the tree key for the given primary key value.
 *
 * @param primaryKeyValue The primary key value.
 * @param virtualTable The virtual table instance.
 * @return The tree key.
 */
static Byte* createTreeKey(sqlite3_value* primaryKeyValue,
btreesModsVirtualTable* virtualTable);

/**
 * Creates the tree key for the given row id.
 *
 * @param rowId The row id.
 * @param virtualTable The virtual table instance.
 * @return The tree key.
 */
static Byte* createTreeKey(sqlite_int64 rowId, btreesModsVirtualTable*
virtualTable);

/**
 * Creates the tree key for the given primary key value and row id.
 *
 * @param primaryKeyValue The primary key value.
 * @param rowId The row id.
 * @param virtualTable The virtual table instance.
 * @return The tree key.
 */
static Byte* createTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId, btreesModsVirtualTable* virtualTable);

/**
 * Creates the integer tree key for the given primary key value and row id.
 *
 * @param primaryKeyValue The primary key value.
 * @param rowId The row id.
 * @return The tree key.
 */
static Byte* createIntTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId);

/**
 * Creates the float tree key for the given primary key value and row id.
 *
 * @param primaryKeyValue The primary key value.
 * @param rowId The row id.
 * @return The tree key.

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

*/
static Byte* createFloatTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId);

/**
 * Creates the text tree key for the given primary key value and row id.
 *
 * @param primaryKeyValue The primary key value.
 * @param rowId The row id.
 * @return The tree key.
 */
static Byte* createTextTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId);

/**
 * Creates the blob tree key for the given primary key value and row id.
 *
 * @param primaryKeyValue The primary key value.
 * @param rowId The row id.
 * @return The tree key.
 */
static Byte* createBlobTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId);

/**
 * Converts the SQLite value to the string.
 *
 * @param value The SQLite value.
 * @param pString The pointer for saving the converted string.
 */
static void convertSqliteValueToString(sqlite3_value* value, char** pString);

/**
 * Converts the text SQLite value to the string.
 *
 * @param value The SQLite value.
 * @param pString The pointer for saving the converted string.
 */
static void convertSqliteTextValueToString(sqlite3_value* value, char**
pString);

/**
 * Copies the string from the source to the destination.
 * Frees the destination string's memory if necessary and allocates the
memory for it.
 *
 * @param pDestination The pointer to the destination string.
 * @param source The source string.
 */
static void copyString(char** pDestination, const char* source);

/**
 * Frees the given string's memory.
 *
 * @param pString The given string.
 */
static void freeString(char** pString);

/**

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

* Clears the virtual table's index params and frees their memory.
*
* @param pVTab The virtual table instance.
*/
static void freeParams(sqlite3_vtab* pVTab);

/**
* Rebuilds the tree index if the special conditions are matched.
*
* @param virtualTable The virtual table instance.
*/
static void rebuildIndexIfNecessary(btreesModsVirtualTable* virtualTable);

/**
* Rebuilds the tree index.
*
* @param virtualTable The virtual table instance.
*/
static void rebuildIndex(btreesModsVirtualTable* virtualTable);

/**
* Measures max memory usage during the last query execution.
*/
static void measureMaxMemoryUsage();

/**
* Visualizes the B-tree or its modification used in the virtual table to the
GraphViz DOT file.
*
* @param ctx Context for writing the results.
* @param argc The arguments count.
* @param argv The arguments.
*/
static void btreesModsVisualize(sqlite3_context* ctx, int argc,
sqlite3_value** argv);

/**
* Gets the order for the B-tree or its modification used in the virtual
table.
*
* @param ctx Context for writing the results.
* @param argc The arguments count.
* @param argv The arguments.
*/
static void btreesModsGetTreeOrder(sqlite3_context* ctx, int argc,
sqlite3_value** argv);

/**
* Gets the type of the B-tree or its modification used in the virtual table.
*
* @param ctx Context for writing the results.
* @param argc The arguments count.
* @param argv The arguments.
*/
static void btreesModsGetTreeType(sqlite3_context* ctx, int argc,
sqlite3_value** argv);

/**
* Opens the tree for the given virtual table.

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

*
* @param pTree The pointer to the tree.
* @param db The SQLite DB connection.
* @param tableName The virtual table name.
* @return 1 if the tree is successfully opened, false otherwise.
*/
static int openTreeForTable(FileBaseBTree** pTree, sqlite3* db, const char*
tableName, int& dataType);

/**
* The btrees_mods SQLite module.
*/
static sqlite3_module btreesModsModule = {
    0, // iVersion
    btreesModsCreate,
    btreesModsConnect,
    btreesModsBestIndex,
    btreesModsDisconnect,
    btreesModsDestroy,
    btreesModsOpen,
    btreesModsClose,
    btreesModsFilter,
    btreesModsNext,
    btreesModsEof,
    btreesModsColumn,
    btreesModsRowid,
    btreesModsUpdate,
    NULL, // xBegin
    NULL, // xSync
    NULL, // xCommit
    NULL, // xRollback
    NULL, // xFindFunction
    btreesModsRename,
    NULL, // xSavepoint
    NULL, // xRelease
    NULL, // xRollbackTo
    NULL // xShadowName
};

#endif //BTREES_BTREES_MODS_H

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

4. Файл btrees_mods.c

Данный файл является файлом с реализацией методов, описанных в файле btrees_mods.h — заголовочным файле для расширения для SQLite.

```

/// \file
/// \brief      B-tree and modifications SQLite extension.
/// \authors    Anton Rigin
/// \version    0.1.0
/// \date       03.01.2019 -- 04.05.2019
///            The bachelor thesis of Anton Rigin,
///            the HSE Software Engineering 4-th year bachelor student.
///
////////////////////////////////////
///

#include "btrees_mods.h"

static int btreesModsCreate(sqlite3* db, void* pAux, int argc, const char*
const* argv,
    sqlite3_vtab** ppVTab, char** pErr)
{
    return btreesModsInit(db, pAux, argc, argv, ppVTab, pErr, TRUE);
}

static int btreesModsConnect(sqlite3* db, void* pAux, int argc, const char*
const* argv,
    sqlite3_vtab** ppVTab, char** pErr)
{
    return btreesModsInit(db, pAux, argc, argv, ppVTab, pErr, FALSE);
}

static int btreesModsBestIndex(sqlite3_vtab* tab, sqlite3_index_info*
pIdxInfo)
{
    int rc = SQLITE_OK;

    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) tab;
    rebuildIndexIfNecessary(virtualTable);
    pIdxInfo->idxNum = virtualTable->params.bestIndex;

    pIdxInfo->idxStr = (char*) sqlite3_malloc(2 * pIdxInfo->nConstraint);

    int i = 0;
    for ( ; i < pIdxInfo->nConstraint; ++i)
    {
        pIdxInfo->aConstraintUsage[i].argvIndex = i + 1;
        pIdxInfo->aConstraintUsage[i].omit = pIdxInfo->aConstraint[i].usable;

        pIdxInfo->idxStr[2 * i] = pIdxInfo->aConstraint[i].iColumn;
        pIdxInfo->idxStr[2 * i + 1] = pIdxInfo->aConstraint[i].op;
    }

    pIdxInfo->needToFreeIdxStr = TRUE;

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    return rc;
}

static int btreesModsDisconnect(sqlite3_vtab* pVtab)
{
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVtab;
    close(&virtualTable->tree);
    freeParams(pVtab);
    return SQLITE_OK;
}

static int btreesModsDestroy(sqlite3_vtab* pVtab)
{
    int rc = SQLITE_OK;

    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVtab;

    sqlite3_str* dropSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(dropSql, "DROP TABLE %s_real;", virtualTable->tableName);
    rc = executeSqlAndFinalize(virtualTable->db,
    sqlite3_str_finish(dropSql));

    if (rc)
        return rc;

    sqlite3_str* deleteSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(deleteSql, "DELETE FROM btrees_mods_idxinfo WHERE
    tableName = \"%s\";", virtualTable->tableName);
    rc = executeSqlAndFinalize(virtualTable->db,
    sqlite3_str_finish(deleteSql));

    if (rc)
        return rc;

    close(&virtualTable->tree);
    remove(virtualTable->params.treeFileName);
    freeParams(pVtab);

    return SQLITE_OK;
}

static int btreesModsFilter(sqlite3_vtab_cursor* cursor, int idxNum, const
char* idxStr,
    int argc, sqlite3_value** argv)
{
    int rc = SQLITE_OK;

    btreesModsCursor* customCursor = (btreesModsCursor*) cursor;
    customCursor->rowsIdsCount = -1;
    customCursor->currentRowIdIdx = -1;

    int i = 0;
    for ( ; i < argc; ++i)
    {
        int columnNum = idxStr[2 * i];
        unsigned char operation = idxStr[2 * i + 1];

        rc = btreesModsHandleConstraint(customCursor, columnNum, operation,

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

argv[i]);

    if (customCursor->currentRowIdIdx >= 0)
        return rc;

    if (customCursor->rowsIdsCount >= 0)
        return rc;

    if (rc)
        return rc;
}

printf("No primary key value given in the query\n");

return SQLITE_MISUSE;
}

static int btreesModsNext(sqlite3_vtab_cursor* cursor)
{
    int rc = SQLITE_OK;

    btreesModsCursor* customCursor = (btreesModsCursor*) cursor;

    if (customCursor->currentRowIdIdx == ROWID_IDX_EOF)
        return rc;

    if (customCursor->currentRowIdIdx == customCursor->rowsIdsCount - 1)
    {
        customCursor->currentRowIdIdx = ROWID_IDX_EOF;
        return rc;
    }

    ++customCursor->currentRowIdIdx;

    return rc;
}

static int btreesModsEof(sqlite3_vtab_cursor* cursor)
{
    btreesModsCursor* customCursor = (btreesModsCursor*) cursor;
    return customCursor->currentRowIdIdx == ROWID_IDX_EOF;
}

static int btreesModsColumn(sqlite3_vtab_cursor* cursor, sqlite3_context*
context, int n)
{
    int rc = SQLITE_OK;

    btreesModsCursor* customCursor = (btreesModsCursor*) cursor;
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) cursor-
>pVtab;

    sqlite3_str* pSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(pSql, "SELECT * FROM %s_real WHERE rowid = %d;",
virtualTable->tableName,
        customCursor->rowsIds[customCursor->currentRowIdIdx]);
    sqlite3_stmt* stmt = NULL;

    rc = executeSql(virtualTable->db, sqlite3_str_finish(pSql), &stmt);

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    if (rc)
        return rc;

    sqlite3_result_value(context, sqlite3_column_value(stmt, n));

    rc = sqlite3_finalize(stmt);

    return rc;
}

static int btreesModsHandleConstraint(btreesModsCursor* cursor, int
columnNum, unsigned char operation,
    sqlite3_value* exprValue)
{
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) cursor-
>base.pVtab;

    if (columnNum == virtualTable->params.indexColNumber)
    {
        switch (operation)
        {
            case SQLITE_INDEX_CONSTRAINT_EQ:
                return btreesModsHandleConstraintEq(cursor, exprValue);
            default:
                printf("Only equality comparing is supported\n");
                return SQLITE_MISUSE;
        }
    }
}

static int btreesModsHandleConstraintEq(btreesModsCursor* cursor,
sqlite3_value* exprValue)
{
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) cursor-
>base.pVtab;
    Byte* searchedKey = createTreeKey(exprValue, virtualTable);
    byteComparator.firstPartBytes = virtualTable->params.indexDataSize;

    Byte** keys = NULL;
    int keysCount = searchAll(virtualTable->tree, searchedKey, &keys);
    ++virtualTable->stats.searchesCount;

    cursor->rowsIdsCount = keysCount;

    if (keysCount == 0)
        return SQLITE_OK;

    cursor->rowsIds = (sqlite_int64*) malloc(keysCount *
sizeof(sqlite_int64));

    for (int i = 0; i < keysCount; ++i)
    {
        memcpy(&cursor->rowsIds[i], &keys[i][virtualTable-
>params.indexDataSize], sizeof(sqlite_int64));
        free(keys[i]);
    }

    cursor->currentRowIdIdx = 0;
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    free(keys);

    return SQLITE_OK;
}

static int btreesModsUpdate(sqlite3_vtab* pVTab, int argc, sqlite3_value**
argv, sqlite_int64* pRowid)
{
    if (argc == 1 && sqlite3_value_type(argv[0]) != SQLITE_NULL)
        return btreesModsDelete(pVTab, argv[0], pRowid);
    else if (argc > 1 && sqlite3_value_type(argv[0]) == SQLITE_NULL)
        return btreesModsInsert(pVTab, argc, argv, pRowid);
    else if (argc > 1 && sqlite3_value_type(argv[0]) != SQLITE_NULL)
        return btreesModsDoUpdate(pVTab, argc, argv, pRowid);
    else
        return -1;
}

static int btreesModsDoUpdate(sqlite3_vtab* pVTab, int argc, sqlite3_value**
argv, sqlite_int64* pRowid)
{
    int rc = SQLITE_OK;
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVTab;

    if (sqlite3_value_type(argv[virtualTable->params.indexColNumber + 2]) !=
        getIntByDataType(virtualTable->params.indexDataType))
        return SQLITE_MISMATCH;

    sqlite3_str* selectSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(selectSql, "SELECT * FROM %s_real;", virtualTable-
>tableName);
    sqlite3_stmt* selectStmt = NULL;
    rc = executeSql(virtualTable->db, sqlite3_str_finish(selectSql),
    &selectStmt);

    if (rc)
        return rc;

    sqlite3_str* pSql = sqlite3_str_new(virtualTable->db);
    char* strValue = NULL;
    convertSqliteValueToString(argv[2], &strValue);
    sqlite3_str_appendf(pSql, "UPDATE %s_real SET %s = %s", virtualTable-
>tableName,
        sqlite3_column_name(selectStmt, 0), strValue);

    int i = 3;
    for (; i < argc; ++i)
    {
        convertSqliteValueToString(argv[i], &strValue);
        sqlite3_str_appendf(pSql, ", %s = %s",
sqlite3_column_name(selectStmt, i - 2), strValue);
    }

    convertSqliteValueToString(argv[0], &strValue);
    sqlite3_str_appendf(pSql, " WHERE %s = %s;", virtualTable-
>params.indexColName, strValue);

    free(strValue);

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

rc = sqlite3_finalize(selectStmt);

if (rc)
    return rc;

rc = executeSqlAndFinalize(virtualTable->db, sqlite3_str_finish(pSql));

if (rc)
    return rc;

*pRowid = getRowId(virtualTable, argv[virtualTable->params.indexColNumber
+ 2]);

if (strcmp((char*) sqlite3_value_text(argv[0]),
          (char*) sqlite3_value_text(argv[virtualTable-
>params.indexColNumber + 2])) != 0)
{
    byteComparator.firstPartBytes = virtualTable->params.indexDataSize;

    rc = (int) !removeKey(virtualTable->tree, createTreeKey(argv[0],
virtualTable));
    ++virtualTable->stats.deletesCount;

    if (rc)
        return rc;

    insert(virtualTable->tree, createTreeKey(argv[virtualTable-
>params.indexColNumber + 2],
        *pRowid, virtualTable));
    ++virtualTable->stats.insertsCount;
}

return rc;
}

static int btreesModsDelete(sqlite3_vtab* pVTab, sqlite3_value*
primaryKeyValue, sqlite_int64* pRowid)
{
    int rc = SQLITE_OK;
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVTab;
    sqlite3_str* pSql = sqlite3_str_new(virtualTable->db);
    char* strValue = NULL;
    convertSqliteValueToString(primaryKeyValue, &strValue);
    sqlite3_str_appendf(pSql, "DELETE FROM %s_real WHERE %s = %s;",
        virtualTable->tableName,
        virtualTable->params.indexColName,
        strValue);
    free(strValue);
    rc = executeSqlAndFinalize(virtualTable->db, sqlite3_str_finish(pSql));

    if (rc)
        return rc;

    byteComparator.firstPartBytes = virtualTable->params.indexDataSize;

    rc = (int) !removeKey(virtualTable->tree, createTreeKey(primaryKeyValue,
virtualTable));
    ++virtualTable->stats.deletesCount;
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    *pRowid = getRowId(virtualTable, primaryKeyValue);

    return rc;
}

static int btreesModsInsert(sqlite3_vtab* pVTab, int argc, sqlite3_value**
argv, sqlite_int64* pRowid)
{
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVTab;
    rebuildIndexIfNecessary(virtualTable);

    if (sqlite3_value_type(argv[virtualTable->params.indexColNumber + 2]) !=
        getIntByDataType(virtualTable->params.indexDataType))
        return SQLITE_MISMATCH;

    sqlite3_str* pSql = sqlite3_str_new(virtualTable->db);
    char* strValue = NULL;
    convertSqliteValueToString(argv[2], &strValue);
    sqlite3_str_appendf(pSql, "INSERT INTO %s_real VALUES (%s", virtualTable-
>tableName, strValue);

    int i = 3;
    for ( ; i < argc; ++i)
    {
        convertSqliteValueToString(argv[i], &strValue);
        sqlite3_str_appendf(pSql, ", %s", strValue);
    }

    sqlite3_str_appendf(pSql, ");");

    int rc = executeSqlAndFinalize(virtualTable->db,
sqlite3_str_finish(pSql));

    if (rc)
        return rc;

    *pRowid = getRowId(virtualTable, argv[virtualTable->params.indexColNumber
+ 2]);

    byteComparator.firstPartBytes = virtualTable->params.indexDataSize;

    Byte* treeKey = createTreeKey(argv[virtualTable->params.indexColNumber +
2], *pRowid, virtualTable);

    insert(virtualTable->tree, treeKey);
    ++virtualTable->stats.insertsCount;

    return rc;
}

static int btreesModsOpen(sqlite3_vtab* pVTab, sqlite3_vtab_cursor**
ppCursor)
{
    btreesModsCursor* cursor = (btreesModsCursor*)
sqlite3_malloc(sizeof(btreesModsCursor));
    memset(cursor, 0, sizeof(btreesModsCursor));
    cursor->base.pVtab = pVTab;
    cursor->currentRowIdIdx = ROWID_IDX_EOF;
    *ppCursor = (sqlite3_vtab_cursor*) cursor;
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    return SQLITE_OK;
}

static int btreesModsClose(sqlite3_vtab_cursor* pCur)
{
    sqlite3_free(pCur);
    return SQLITE_OK;
}

static int btreesModsRowid(sqlite3_vtab_cursor *pCur, sqlite_int64 *pRowid)
{
    btreesModsCursor* cursor = (btreesModsCursor*) pCur;

    if (cursor->currentRowIdIdx == ROWID_IDX_EOF)
        pRowid = NULL;
    else
        *pRowid = cursor->rowsIds[cursor->currentRowIdIdx];

    return SQLITE_OK;
}

static int btreesModsRename(sqlite3_vtab* pVtab, const char* zNew)
{
    int rc = SQLITE_OK;

    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVtab;

    sqlite3_str* renameSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(renameSql, "ALTER TABLE %s_real RENAME TO %s_real;",
        virtualTable->tableName, zNew);
    rc = executeSqlAndFinalize(virtualTable->db,
        sqlite3_str_finish(renameSql));

    if (rc)
        return rc;

    sqlite3_str* updateSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(updateSql, "UPDATE btrees_mods_idxinfo SET tableName
= \"%s\" WHERE tableName = \"%s\";",
        zNew, virtualTable->tableName);
    rc = executeSqlAndFinalize(virtualTable->db,
        sqlite3_str_finish(updateSql));

    if (rc)
        return rc;

    if (virtualTable->tableName)
    {
        free(virtualTable->tableName);
        virtualTable->tableName = NULL;
    }

    size_t newStrLen = strlen(zNew);
    virtualTable->tableName = (char*) malloc(newStrLen);
    strcpy(virtualTable->tableName, zNew);

    return rc;
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата


```

static int btreesModsInit(sqlite3* db, void* pAux, int argc, const char*
const* argv,
    sqlite3_vtab** ppVTab, char** pzErr, int isCreate)
{
    int rc = SQLITE_OK;

    srand(time(NULL));

    sqlite3_str* realSql = sqlite3_str_new(db);
    sqlite3_str* virtualSql = sqlite3_str_new(db);
    sqlite3_str_appendf(realSql, "CREATE TABLE %s_real(%s", argv[2],
argv[3]);
    sqlite3_str_appendf(virtualSql, "CREATE TABLE %s(%s", argv[2], argv[3]);

    int i = 4;
    for (; i < argc; ++i)
    {
        sqlite3_str_appendf(realSql, ", %s", argv[i]);
        sqlite3_str_appendf(virtualSql, ", %s", argv[i]);
    }

    sqlite3_str_appendf(realSql, ");");
    sqlite3_str_appendf(virtualSql, ");");
    char* zRealSql = sqlite3_str_finish(realSql);
    char* zVirtualSql = sqlite3_str_finish(virtualSql);

    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*)
sqlite3_malloc(sizeof(btreesModsVirtualTable));
    virtualTable->db = NULL;
    virtualTable->tree = NULL;
    virtualTable->tableName = NULL;
    virtualTable->params.indexColName = NULL;
    virtualTable->params.indexDataType = NULL;
    virtualTable->params.treeFileName = NULL;
    virtualTable->stats.searchesCount = 0;
    virtualTable->stats.insertsCount = 0;
    virtualTable->stats.deletesCount = 0;
    virtualTable->stats.isOriginalStats = FALSE;

    if (isCreate)
    {
        virtualTable->stats.isOriginalStats = TRUE;

        rc = executeSqlAndFinalize(db, zRealSql);

        if (rc)
        {
            *pzErr = sqlite3_mprintf("%s", sqlite3_errmsg(db));
            return rc;
        }

        sqlite3_str* selectSql = sqlite3_str_new(db);
        sqlite3_str_appendf(selectSql, "SELECT * FROM %s_real;", argv[2]);
        char* selectZSql = sqlite3_str_finish(selectSql);
        sqlite3_stmt* selectStmt = NULL;
        rc = sqlite3_prepare_v2(db, selectZSql, -1, &selectStmt, 0);
        *pzErr = sqlite3_mprintf("%s", sqlite3_errmsg(db));

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

if (rc)
{
    *pzErr = sqlite3_mprintf("%s", sqlite3_errmsg(db));
    return rc;
}

char treeFileName[CHAR_BUFFER_SIZE];
char* filledTreeFileName = getTreeFileName(treeFileName);

rc = registerIndexColumn(db, selectStmt, virtualTable, argv[2],
filledTreeFileName);

if (rc)
{
    *pzErr = sqlite3_mprintf("%s", sqlite3_errmsg(db));
    return rc;
}

rc = createIndex(virtualTable, TREE_ORDER, virtualTable-
>params.indexDataSize + sizeof(sqlite_int64));

if (rc)
{
    *pzErr = sqlite3_mprintf("Cannot open tree file for writing or
invalid tree type");
    sqlite3_finalize(selectStmt);
    return rc;
}

sqlite3_finalize(selectStmt);
}
else
{
    sqlite3_str* getParamsSql = sqlite3_str_new(db);
    sqlite3_str_appendf(getParamsSql, "SELECT * FROM btrees_mods_idxinfo
WHERE tableName = \"%s\"", argv[2]);
    sqlite3_stmt* getParamsStmt = NULL;
    executeSql(db, sqlite3_str_finish(getParamsSql), &getParamsStmt);

    int colNum = 1;

    virtualTable->params.bestIndex = sqlite3_column_int(getParamsStmt,
colNum++);
    virtualTable->params.indexColNumber =
sqlite3_column_int(getParamsStmt, colNum++);
    copyString(&virtualTable->params.indexColName, (char*)
sqlite3_column_text(getParamsStmt, colNum++));
    copyString(&virtualTable->params.indexDataType, (char*)
sqlite3_column_text(getParamsStmt, colNum++));
    virtualTable->params.indexDataSize =
sqlite3_column_int(getParamsStmt, colNum++);
    copyString(&virtualTable->params.treeFileName, (char*)
sqlite3_column_text(getParamsStmt, colNum++));

    rc = openIndex(virtualTable);

    if (rc)
    {
        *pzErr = sqlite3_mprintf("Cannot open tree file for reading or

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

invalid tree type");
    sqlite3_finalize(getParamsStmt);
    return rc;
}

sqlite3_finalize(getParamsStmt);
}

rc = sqlite3_declare_vtab(db, zVirtualSql);

if (rc)
{
    *pzErr = sqlite3_mprintf("%s", sqlite3_errmsg(db));
    return rc;
}

virtualTable->base.pModule = &btreesModsModule;
virtualTable->base.nRef = 1;
virtualTable->base.zErrMsg = NULL;
virtualTable->db = db;
size_t tableNameStrLen = strlen(argv[2]);
virtualTable->tableName = (char*) malloc(tableNameStrLen);
strcpy(virtualTable->tableName, argv[2]);
*ppVTab = (sqlite3_vtab*) virtualTable;

sqlite3_free(zRealSql);
sqlite3_free(zVirtualSql);

return rc;
}

static char* getTreeFileName(char* treeFileName)
{
    const char* treePrefix = "tree_";
    strcpy(treeFileName, treePrefix);

    char treeRandomId[CHAR_BUFFER_SIZE];
    sprintf(treeRandomId, "%d", rand());
    strcat(treeFileName, treeRandomId);

    char treeTimeStamp[CHAR_BUFFER_SIZE];
    sprintf(treeTimeStamp, "%d", (int) time(NULL));
    strcat(treeFileName, treeTimeStamp);

    const char* treeFileExtension = ".btree";
    strcat(treeFileName, treeFileExtension);

    return treeFileName;
}

static int createIndex(btreesModsVirtualTable* virtualTable, int order, int
keySize)
{
    switch (virtualTable->params.bestIndex)
    {
        case BTREE_NUM:
            create(&virtualTable->tree, BaseBTree::TreeType::B_TREE,
                order, keySize, virtualTable->params.treeFileName);
            break;
    }
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    case BPLUSTREE_NUM:
        create(&virtualTable->tree, BaseBTree::TreeType::B_PLUS_TREE,
            order, keySize, virtualTable->params.treeFileName);
        break;
    case BSTARTREE_NUM:
        create(&virtualTable->tree, BaseBTree::TreeType::B_STAR_TREE,
            order, keySize, virtualTable->params.treeFileName);
        break;
    case BSTARPLUSTREE_NUM:
        create(&virtualTable->tree,
BaseBTree::TreeType::B_STAR_PLUS_TREE,
            order, keySize, virtualTable->params.treeFileName);
        break;
    default:
        return ERROR_CODE;
}

if (virtualTable->tree)
    return SQLITE_OK;
else
    return ERROR_CODE;
}

static int openIndex(btreesModsVirtualTable* virtualTable)
{
    switch (virtualTable->params.bestIndex)
    {
        case BTREE_NUM:
            open(&virtualTable->tree, BaseBTree::TreeType::B_TREE,
virtualTable->params.treeFileName);
            break;
        case BPLUSTREE_NUM:
            open(&virtualTable->tree, BaseBTree::TreeType::B_PLUS_TREE,
virtualTable->params.treeFileName);
            break;
        case BSTARTREE_NUM:
            open(&virtualTable->tree, BaseBTree::TreeType::B_STAR_TREE,
virtualTable->params.treeFileName);
            break;
        case BSTARPLUSTREE_NUM:
            open(&virtualTable->tree, BaseBTree::TreeType::B_STAR_PLUS_TREE,
virtualTable->params.treeFileName);
            break;
        default:
            return ERROR_CODE;
    }

    if (virtualTable->tree)
        return SQLITE_OK;
    else
        return ERROR_CODE;
}

static int registerIndexColumn(sqlite3* db, sqlite3_stmt* stmt,
btreesModsVirtualTable* virtualTable,
    const char* tableName, const char* treeFileName)
{
    int rc = SQLITE_OK;

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

const char* columnName = NULL;

const char* dataType = NULL;
const char* collSeq = NULL;

int notNull, primaryKey, autoInc;

sqlite3_str* createSql = sqlite3_str_new(db);
sqlite3_str_appendf(createSql, "CREATE TABLE IF NOT EXISTS
btrees_mods_idxinfo("
                        "tableName TEXT PRIMARY KEY, "
                        "bestIndex INTEGER, "
                        "indexColNumber INTEGER, "
                        "indexColName TEXT, "
                        "indexDataType TEXT, "
                        "indexDataSize INTEGER, "
                        "treeFileName TEXT);");

rc = executeSqlAndFinalize(db, sqlite3_str_finish(createSql));

if (rc)
    return rc;

virtualTable->params.indexColNumber = -1;
copyString(&virtualTable->params.indexColName, "rowid");
copyString(&virtualTable->params.indexDataType, "INTEGER");
virtualTable->params.indexDataSize = getDataSizeByType(virtualTable-
>params.indexDataType);
virtualTable->params.bestIndex = BPLUSTREE_NUM;
copyString(&virtualTable->params.treeFileName, treeFileName);

int i = 0;
for ( ; (columnName = sqlite3_column_name(stmt, i)) != NULL; ++i)
{
    rc = sqlite3_table_column_metadata(db, NULL,
sqlite3_mprintf("%s_real", tableName), columnName,
                &dataType, &collSeq, &notNull, &primaryKey, &autoInc);

    if (rc)
        return rc;

    if (primaryKey)
    {
        virtualTable->params.indexColNumber = i;
        copyString(&virtualTable->params.indexColName, columnName);
        copyString(&virtualTable->params.indexDataType, dataType);
        virtualTable->params.indexDataSize = getDataSizeByType(dataType);

        break;
    }
}

sqlite3_str* insertSql = sqlite3_str_new(db);
sqlite3_str_appendf(insertSql, "INSERT INTO btrees_mods_idxinfo "
                        "VALUES (\"%s\", %d, %d, \"%s\", \"%s\",
%d, \"%s\");",
                        tableName,
                        virtualTable->params.bestIndex,
                        virtualTable->params.indexColNumber,
                        virtualTable->params.indexColName,

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

virtualTable->params.indexDataType,
virtualTable->params.indexDataSize,
virtualTable->params.treeFileName);
rc = executeSqlAndFinalize(db, sqlite3_str_finish(insertSql));

return rc;
}

static int getDataSizeByType(const char* dataType)
{
    if (strcmp(dataType, "INTEGER") == 0)
        return INTEGER_SIZE;
    else if (strcmp(dataType, "FLOAT") == 0)
        return FLOAT_SIZE;
    else if (strcmp(dataType, "TEXT") == 0)
        return TEXT_SIZE;
    else if (strcmp(dataType, "BLOB") == 0)
        return BLOB_SIZE;
    else if (strcmp(dataType, "NULL") == 0)
        return NULL_SIZE;
    else
        return ERROR_CODE;
}

static const char* getDataTypeByInt(int dataType)
{
    switch (dataType)
    {
        case SQLITE_INTEGER:
            return "INTEGER";
        case SQLITE_FLOAT:
            return "FLOAT";
        case SQLITE_TEXT:
            return "TEXT";
        case SQLITE_BLOB:
            return "BLOB";
        case SQLITE_NULL:
            return "NULL";
        default:
            return NULL;
    }
}

static int getIntByDataType(const char* dataType)
{
    if (strcmp(dataType, "INTEGER") == 0)
        return SQLITE_INTEGER;
    else if (strcmp(dataType, "FLOAT") == 0)
        return SQLITE_FLOAT;
    else if (strcmp(dataType, "TEXT") == 0)
        return SQLITE_TEXT;
    else if (strcmp(dataType, "BLOB") == 0)
        return SQLITE_BLOB;
    else if (strcmp(dataType, "NULL") == 0)
        return SQLITE_NULL;
    else
        return -1;
}

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

static sqlite3_int64 getId(btreesModsVirtualTable* virtualTable,
sqlite3_value* primaryKeyValue)
{
    char* strValue = NULL;

    sqlite3_str* selectSql = sqlite3_str_new(virtualTable->db);
    convertSqliteValueToString(primaryKeyValue, &strValue);
    sqlite3_str_appendf(selectSql, "SELECT rowid FROM %s_real WHERE %s = %s;",
                                virtualTable->tableName,
                                virtualTable->params.indexColName,
                                strValue);

    free(strValue);
    sqlite3_stmt* stmt = NULL;
    executeSql(virtualTable->db, sqlite3_str_finish(selectSql), &stmt);

    sqlite3_int64 rowId = sqlite3_column_int64(stmt, 0);

    sqlite3_finalize(stmt);

    return rowId;
}

static int executeSqlAndFinalize(sqlite3* db, char* sql)
{
    sqlite3_stmt* stmt = NULL;
    int rc = executeSql(db, sql, &stmt);
    sqlite3_finalize(stmt);
    return rc;
}

static int executeSql(sqlite3* db, char* sql, sqlite3_stmt** stmt)
{
    int rc = SQLITE_OK;

    if (!sql)
        rc = SQLITE_NOMEM;
    else if (SQLITE_OK == (rc = sqlite3_prepare_v2(db, sql, -1, stmt, 0)))
        rc = sqlite3_step(*stmt);

    rc = rc == SQLITE_DONE || rc == SQLITE_ROW ? SQLITE_OK : rc;

    return rc;
}

static Byte* createTreeKey(sqlite3_value* primaryKeyValue,
btreesModsVirtualTable* virtualTable)
{
    return createTreeKey(primaryKeyValue, 0, virtualTable);
}

static Byte* createTreeKey(sqlite_int64 rowId, btreesModsVirtualTable*
virtualTable)
{
    return createTreeKey(NULL, rowId, virtualTable);
}

static Byte* createTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId, btreesModsVirtualTable* virtualTable)

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

{
    if (primaryKeyValue == NULL)
    {
        Byte* treeKey = (Byte*) malloc(virtualTable->params.indexDataSize +
sizeof(sqlite_int64));

        memcpy(treeKey, (Byte*) &rowId, virtualTable->params.indexDataSize);
        memcpy(&treeKey[virtualTable->params.indexDataSize], (Byte*) &rowId,
sizeof(sqlite_int64));

        return treeKey;
    }
    else
    {
        int dataType = sqlite3_value_type(primaryKeyValue);

        switch (dataType)
        {
            case SQLITE_INTEGER:
                return createIntTreeKey(primaryKeyValue, rowId);
            case SQLITE_FLOAT:
                return createFloatTreeKey(primaryKeyValue, rowId);
            case SQLITE_TEXT:
                return createTextTreeKey(primaryKeyValue, rowId);
            case SQLITE_BLOB:
                return createBlobTreeKey(primaryKeyValue, rowId);
            case SQLITE_NULL:
                return NULL;
        }
    }
}

static Byte* createIntTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId)
{
    Byte* treeKey = (Byte*) malloc(INTEGER_SIZE + sizeof(sqlite_int64));

    int value = sqlite3_value_int(primaryKeyValue);
    memcpy(treeKey, (Byte*) &value, INTEGER_SIZE);
    memcpy(&treeKey[INTEGER_SIZE], (Byte*) &rowId, sizeof(sqlite_int64));

    return treeKey;
}

static Byte* createFloatTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId)
{
    Byte* treeKey = (Byte*) malloc(FLOAT_SIZE + sizeof(sqlite_int64));

    double value = sqlite3_value_double(primaryKeyValue);
    memcpy(treeKey, (Byte*) &value, FLOAT_SIZE);
    memcpy(&treeKey[FLOAT_SIZE], (Byte*) &rowId, sizeof(sqlite_int64));

    return treeKey;
}

static Byte* createTextTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId)
{

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата


```

int size = sqlite3_value_bytes(primaryKeyValue);
size = size >= TEXT_SIZE ? TEXT_SIZE : size;

Byte* treeKey = (Byte*) malloc(TEXT_SIZE + sizeof(sqlite_int64));
memset(treeKey, 0, TEXT_SIZE + sizeof(sqlite_int64));

memcpy(treeKey, (Byte*) sqlite3_value_text(primaryKeyValue), size);
memcpy(&treeKey[TEXT_SIZE], (Byte*) &rowId, sizeof(sqlite_int64));

return treeKey;
}

static Byte* createBlobTreeKey(sqlite3_value* primaryKeyValue, sqlite_int64
rowId)
{
    int size = sqlite3_value_bytes(primaryKeyValue);
    size = size >= BLOB_SIZE ? BLOB_SIZE : size;

    Byte* treeKey = (Byte*) malloc(BLOB_SIZE + sizeof(sqlite_int64));
    memset(treeKey, 0, BLOB_SIZE + sizeof(sqlite_int64));

    memcpy(treeKey, (Byte*) sqlite3_value_blob(primaryKeyValue), size);
    memcpy(&treeKey[BLOB_SIZE], (Byte*) &rowId, sizeof(sqlite_int64));

    return treeKey;
}

static void convertSqliteValueToString(sqlite3_value* value, char** pString)
{
    if (*pString)
    {
        free(*pString);
        *pString = NULL;
    }

    *pString = (char*) malloc(TEXT_SIZE);

    switch (sqlite3_value_type(value))
    {
        case SQLITE_INTEGER:
        case SQLITE_FLOAT:
            strcpy(*pString, (const char*) sqlite3_value_text(value));
            break;
        case SQLITE_TEXT:
            convertSqliteTextValueToString(value, pString);
            break;
        case SQLITE_BLOB:
            strcpy(*pString, (const char*) sqlite3_value_blob(value));
            break;
        case SQLITE_NULL:
            sprintf(*pString, "NULL");
            break;
    }
}

static void convertSqliteTextValueToString(sqlite3_value* value, char**
pString)
{
    snprintf(*pString, TEXT_SIZE - 1, "\"%s\"", (const char*)

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

sqlite3_value_text(value));
    strcat(*pString, "\\");
}

static void copyString(char** pDestination, const char* source)
{
    if (*pDestination)
        freeString(pDestination);

    *pDestination = (char*) malloc(strlen(source));
    strcpy(*pDestination, source);
}

static void freeString(char** pString)
{
    free(*pString);
    *pString = NULL;
}

static void freeParams(sqlite3_vtab* pVTab)
{
    btreesModsVirtualTable* virtualTable = (btreesModsVirtualTable*) pVTab;

    free(virtualTable->params.indexColName);
    virtualTable->params.indexColName = NULL;
    free(virtualTable->params.indexDataType);
    virtualTable->params.indexDataType = NULL;
    free(virtualTable->params.treeFileName);
    virtualTable->params.treeFileName = NULL;

    free(virtualTable->tableName);

    sqlite3_free(pVTab);
}

static void rebuildIndexIfNecessary(btreesModsVirtualTable* virtualTable)
{
    int updatesCount = virtualTable->stats.insertsCount + virtualTable-
>stats.deletesCount;
    int totalOperationsCount = updatesCount + virtualTable-
>stats.searchesCount;

    if (!virtualTable->stats.isOriginalStats)
        return;

    if (totalOperationsCount == 0)
        return;

    if (totalOperationsCount > REBUILD_MAX_COUNT)
        return;

    if (totalOperationsCount % REBUILD_COUNT != 0)
        return;

    if (updatesCount < REBUILD_COEF * totalOperationsCount)
        return;

    int oldBestIndex = virtualTable->params.bestIndex;

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

    if (virtualTable->stats.insertsCount > REBUILD_SPLIT_PERCENT_POINT *
updatesCount)
        virtualTable->params.bestIndex = BSTARTREE_NUM;
    else
        virtualTable->params.bestIndex = BSTARPLUSTREE_NUM;

    if (virtualTable->params.bestIndex != oldBestIndex)
        rebuildIndex(virtualTable);
}

static void rebuildIndex(btreesModsVirtualTable* virtualTable)
{
    searchAllByteComparator.firstPartBytes = virtualTable-
>params.indexDataSize;
    setSearchAllByteComparator(virtualTable->tree);
    Byte** keys = NULL;
    int keysCount = searchAll(virtualTable->tree,
createTreeKey((sqlite_int64) 0, virtualTable), &keys);
    setByteComparator(virtualTable->tree);

    close(&virtualTable->tree);
    remove(virtualTable->params.treeFileName);

    createIndex(virtualTable, TREE_ORDER, virtualTable->params.indexDataSize
+ sizeof(sqlite_int64));

    int i = 0;
    for ( ; i < keysCount; ++i)
        insert(virtualTable->tree, keys[i]);

    sqlite3_str* updateParamsSql = sqlite3_str_new(virtualTable->db);
    sqlite3_str_appendf(updateParamsSql, "UPDATE btrees_mods_idxinfo SET
bestIndex = %d WHERE tableName = \"%s\";",
        virtualTable->params.bestIndex, virtualTable->tableName);
    executeSqlAndFinalize(virtualTable->db,
sqlite3_str_finish(updateParamsSql));
}

static void btreesModsVisualize(sqlite3_context* ctx, int argc,
sqlite3_value** argv)
{
    if (argc != 2)
    {
        sqlite3_result_text(ctx, "Arguments count should be equal to 2", -1,
NULL);
        return;
    }

    if (sqlite3_value_type(argv[0]) != SQLITE3_TEXT ||
sqlite3_value_type(argv[1]) != SQLITE3_TEXT)
    {
        sqlite3_result_text(ctx, "Arguments types should be TEXT", -1, NULL);
        return;
    }

    sqlite3* db = sqlite3_context_db_handle(ctx);
    FileBaseBTree* tree = NULL;
    int dataType;
    int isFound = openTreeForTable(&tree, db, (const char*)

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

sqlite3_value_text(argv[0]), dataType);

    if (!isFound)
    {
        sqlite3_result_text(ctx, "Table not found or table metadata is
invalid", -1, NULL);
        return;
    }

    switch (dataType)
    {
        case SQLITE_INTEGER:
            setIntBytePrinter(tree);
            break;
        case SQLITE_FLOAT:
            setFloatBytePrinter(tree);
            break;
        case SQLITE_TEXT:
        case SQLITE_BLOB:
            setBytePrinter(tree);
            break;
        case SQLITE_NULL:
            setNullBytePrinter(tree);
            break;
        default:
            printf("Invalid data type\n");
            break;
    }

    if (visualize(tree, (const char*) sqlite3_value_text(argv[1])))
        sqlite3_result_text(ctx, "File written", -1, NULL);
    else
        printf("Cannot open DOT file for writing\n");
}

static void btreesModsGetTreeOrder(sqlite3_context* ctx, int argc,
sqlite3_value** argv)
{
    if (argc != 1)
    {
        sqlite3_result_text(ctx, "Arguments count should be equal to 1", -1,
NULL);
        return;
    }

    if (sqlite3_value_type(argv[0]) != SQLITE3_TEXT)
    {
        sqlite3_result_text(ctx, "Argument's type should be TEXT", -1, NULL);
        return;
    }

    sqlite3* db = sqlite3_context_db_handle(ctx);
    FileBaseBTree* tree = NULL;
    int dataType;
    int isFound = openTreeForTable(&tree, db, (const char*)
sqlite3_value_text(argv[0]), dataType);

    if (!isFound)
    {

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

        sqlite3_result_text(ctx, "Table not found or table metadata is
invalid or cannot open the tree file", -1, NULL);
        return;
    }

    int treeOrder = getOrder(tree);
    sqlite3_result_int(ctx, treeOrder);
}

static void btreesModsGetTreeType(sqlite3_context* ctx, int argc,
sqlite3_value** argv)
{
    if (argc != 1)
    {
        sqlite3_result_text(ctx, "Arguments count should be equal to 1", -1,
NULL);
        return;
    }

    if (sqlite3_value_type(argv[0]) != SQLITE3_TEXT)
    {
        sqlite3_result_text(ctx, "Argument's type should be TEXT", -1, NULL);
        return;
    }

    sqlite3* db = sqlite3_context_db_handle(ctx);

    sqlite3_str* sql = sqlite3_str_new(db);
    sqlite3_str_appendf(sql, "SELECT bestIndex FROM btrees_mods_idxinfo WHERE
tableName = \"%s\";",
        sqlite3_value_text(argv[0]));
    sqlite3_stmt* stmt = NULL;
    executeSql(db, sqlite3_str_finish(sql), &stmt);

    if (sqlite3_column_type(stmt, 0) == SQLITE_NULL)
    {
        sqlite3_result_text(ctx, "Table not found or table metadata is
invalid or cannot open the tree file", -1, NULL);
        sqlite3_finalize(stmt);
        return;
    }

    int bestIndex = sqlite3_column_int(stmt, 0);

    sqlite3_finalize(stmt);

    sqlite3_result_int(ctx, bestIndex);
}

static int openTreeForTable(FileBaseBTree** pTree, sqlite3* db, const char*
tableName, int& dataType)
{
    sqlite3_str* sql = sqlite3_str_new(db);
    sqlite3_str_appendf(sql,
        "SELECT treeFileName, bestIndex, indexDataType FROM
btrees_mods_idxinfo WHERE tableName = \"%s\";",
        tableName);
    sqlite3_stmt* stmt = NULL;
    executeSql(db, sqlite3_str_finish(sql), &stmt);

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```

if (sqlite3_column_type(stmt, 0) == SQLITE_NULL)
{
    sqlite3_finalize(stmt);
    return FALSE;
}

const char* treeFileName = (const char*) sqlite3_column_text(stmt, 0);
int bestIndex = sqlite3_column_int(stmt, 1);
const char* indexDataType = (const char*) sqlite3_column_text(stmt, 2);

dataType = getIntByDataType(indexDataType);

switch (bestIndex)
{
    case BTREE_NUM:
        open(pTree, BaseBTree::TreeType::B_TREE, treeFileName);
        break;
    case BPLUSTREE_NUM:
        open(pTree, BaseBTree::TreeType::B_PLUS_TREE, treeFileName);
        break;
    case BSTARTREE_NUM:
        open(pTree, BaseBTree::TreeType::B_STAR_TREE, treeFileName);
        break;
    case BSTARPLUSTREE_NUM:
        open(pTree, BaseBTree::TreeType::B_STAR_PLUS_TREE, treeFileName);
        break;
    default:
        printf("Invalid index type: %d\n", bestIndex);
        sqlite3_finalize(stmt);
        return FALSE;
}

sqlite3_finalize(stmt);

if (*pTree)
    return TRUE;
else
    return FALSE;
}

#ifdef __cplusplus
extern "C" {
#endif

int sqlite3_btreesmods_init(sqlite3* db, char** pErrMsg, const
sqlite3_api_routines* pApi) {
    int rc = SQLITE_OK;

    SQLITE_EXTENSION_INIT2(pApi);

    rc = sqlite3_create_function(db, "btreesModsVisualize", 2, SQLITE_UTF8,
0, btreesModsVisualize, 0, 0);

    if (rc)
        return rc;

    rc = sqlite3_create_function(db, "btreesModsGetTreeOrder", 1,
SQLITE_UTF8, 0, btreesModsGetTreeOrder, 0, 0);

```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

```
    if (rc)
        return rc;

    rc = sqlite3_create_function(db, "btreesModsGetTreeType", 1, SQLITE_UTF8,
0, btreesModsGetTreeType, 0, 0);

    if (rc)
        return rc;

    rc = sqlite3_create_module(db, "btrees_mods", &btreesModsModule, 0);

    return rc;
}

#ifdef __cplusplus
}; // extern "C"
#endif
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.02.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата