Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э.Баумана)

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:

Метод фрактального поиска в реляционных базах данных на примере Vertica

Студент ИУ7-81 _____ Е. А. Енин

Руководитель ВКР _____ Э. С. Клышинский

Консультант _____ Ю. М. Гаврилова

Нормоконтролер _____ _____

Москва, 2020

Реферат

Расчетно-пояснительная записка 56 с., 42 рис., 7 табл., 12 источников.

Объектом разработки является программный комплекс, реализующий метод фрактального поиска в реляционной колоночной базе данных.

Целью работы является разработка метода фрактального поиска с выполнением запросов к сжатым данным.

Задачи, поставленные в данной работе:

— провести анализ известных методов сжатия баз данных;
— модифицировать метод фрактального поиска;
— разработать программное обеспечение, реализующее предложенный метод.

Содержание

Введение

Объем информации, который нас окружает, неумолимо растет с течением времени. Исследователи международной компании IDC полагают, что к 2023 году объем данных на цифровых носителях достигнет 11,7 зеттабайт, увеличившись вдвое.

В связи с этим, проблемы поиска и хранения информации набирают свою актуальность и становятся достаточно серьезными в сфере бизнеса. Базы данных многих компаний зачастую состоят из таблиц с огромным числом записей, и поднимают вопрос об их эффективном хранении и использовании.

Существуют алгоритмы интеллектуального анализа баз данных, которые в состоянии искать в них зависимости и закономерности, но они не решают проблему эффективного хранения

В данной работе будет рассмотрен алгоритм фрактального поиска, с помощью которого можно сжимать базы в объеме, и модификация для этого алгоритма, способная искать по сжатым данным

1 Аналитическая часть

Рассмотрим предметную область и основные подходы к решению задачи.

1.1 Предметная область

Основным понятием данной работы, является фрактал, от латинского слова fractus, что в переводе означает сломленный, дробленый[6].

Фрактал - это геометрическая фигура, в которой один и тот же фрагмент повторяется при каждом уменьшении масштаба (Лаверье).

Фрактальной называется структура, состоящая из частей, которые в каком-то смысле подобны целому (Мандельброт).

Фракталы это объекты, которые мы называем неправильными, шероховатыми, пористыми или раздробленными, причем указанными свойствами фракталы обладают в одинаковой степени в любом масштабе (Мандельброт).

Фрактал обладает следующими признаками:

а) самоподобием – или приближенным самоподобием;
б) нетривиальной структурой;
в) дробной размерностью.

Фракталы были описаны в 1977 году французским математиком Бенуа Мандельбротом и с тех пор активно используются в самых разнообразных областях.

Помимо понятия фрактал определяют также понятие предфрактал. Предфрактал – самопободный объект, каждый фрагмент которого повторяется в упрощенном виде при уменьшении масштаба. Большая часть объектов природы является именно предфракталами, а не фракталами, как принято считать.

Существует несколько классификаций фракталов. Самой распространенной классификацией является следующая:

а) Геометрические фракталы – самая известная в силу своей наглядности группа фракталов. Могут строиться как на основе некоторой ломаной линии («двумерные» фракталы), так и на основе некоторой поверхности

(«трехмерные» фракталы). основу фрактала называют генератором и на каждой итерации часть генератора заменяется генератором в масштабе.
Примеры геометрических фракталов: снежинка Коха, Т-квадрат, треугольник Серпинского, дерево Пифагора.

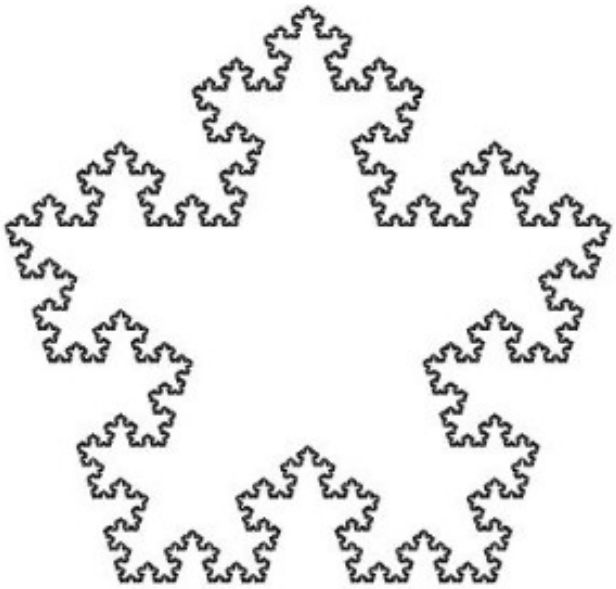

Рисунок 1.1 — Пример геометрического фрактала

б) алгебраические фракталы – самая обширная группа фракталов. алгебраические фракталы получают с помощью нелинейных процессов в n-мерных пространствах.
Пример алгебраического фрактала: множество Мандельброта.

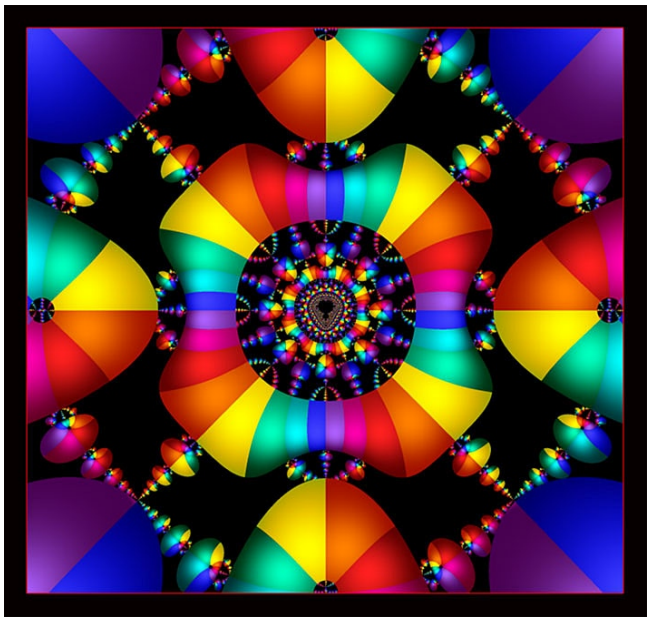

Рисунок 1.2 — Пример алгебраического фрактала

в) Стохастические фракталы. Данные фракталы получаются, если в итерационном процессе менять хаотично некоторые параметры фрактала. объекты, полученные таким образом, похожи на объекты реального мира – например, береговые линии.

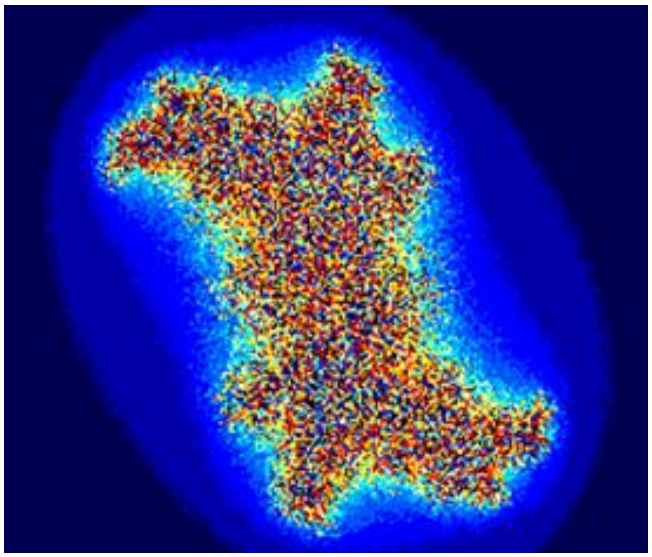

Рисунок 1.3 — Пример стохастического фрактала

По формальному признаку фракталы бывают:

а) связанные;

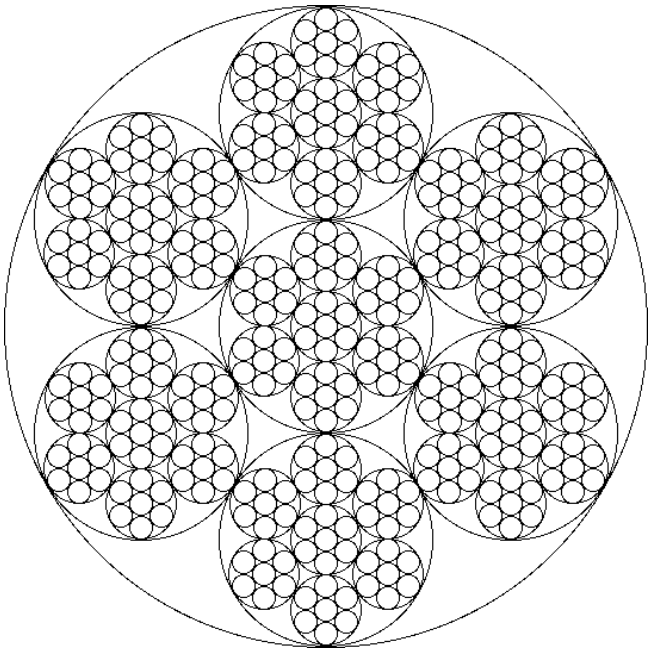

Рисунок 1.4 — Пример связанного фрактала

б) несвязанные.

Рисунок 1.5 — Пример несвязанного фрактала

По размерности фракталы бывают:

а) с целой размерностью;

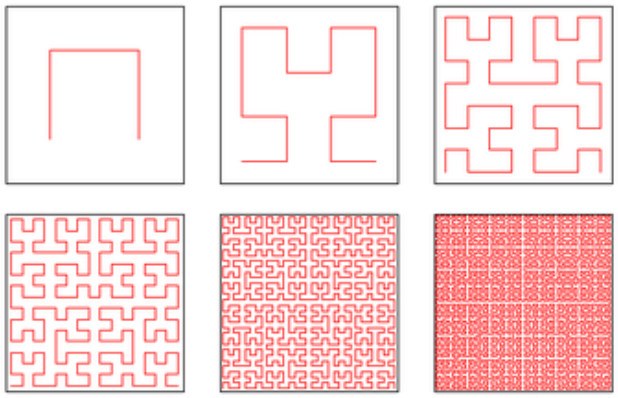

Рисунок 1.6 — Кривая Пеано

б) с дробной размерностью.

Помимо этого фракталы так же делятся на недетерминированные, к которым относятся геометрические и алгебраические фракталы, и на детерминированные (стохастические и алеаторные), линейные, строящиеся по линейному алгоритму, и нелинейные.

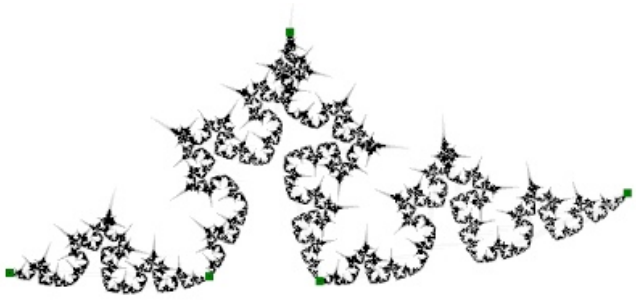

Рисунок 1.7 — фрактал Хельге фон Коха

В виду своей наглядности фракталы нашли свое применение в области компьютерной графики, но их так же можно применять и для баз данных.

Современные базы данных хранят огромное количество записей, описывающих те или иные объекты. Как правило, эти данные имеют некоторую структуру, которую мы можем использовать в своих целях.
В базах данных в качестве самоподобной части фрактала выступают домены.

1.2 Data Mining

Data mining – собирательное название совокупности методов обнаружения в данных «скрытых» закономерностей и дальнейшей работы с ними.

Основные задачи, решаемые с помощью data mining:

а) классификация – отнесение объекта к одному из непересекающихся множеств;

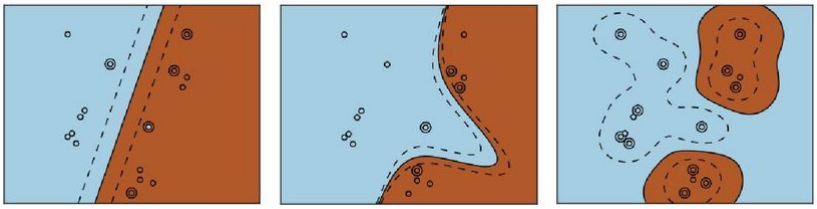

Рисунок 1.8 — Классификация

б) регрессия – установление зависимости непрерывных выходных данных от входных;

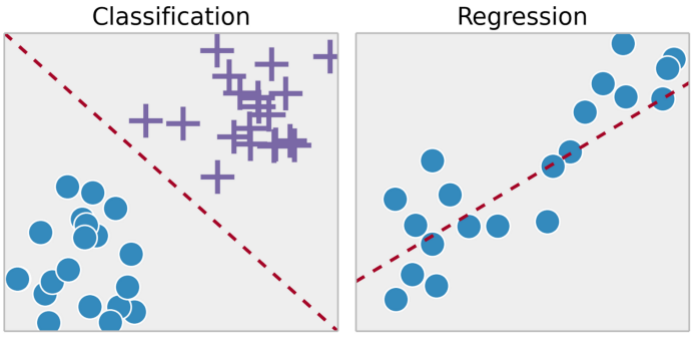

Рисунок 1.9 — регрессия

в) кластеризация – группировка объектов на основе их свойств;

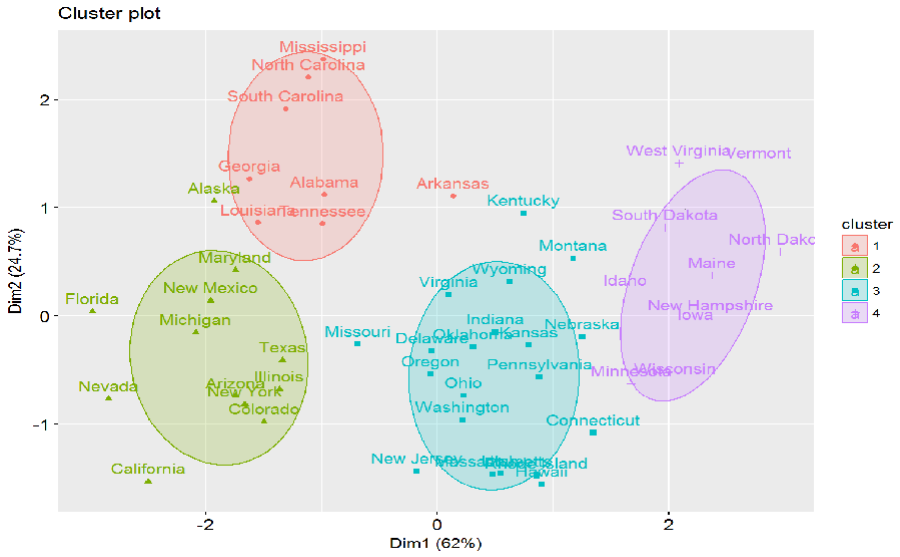

Рисунок 1.10 — Кластеризация

г) ассоциация – выявление закономерностей между связанными событиями;

д) прогнозирование – оценка на основе уже имеющихся данных пропущенных или будущих значений;

е) определение отклонений – обнаружение данных, наиболее отличающихся от общего множества.

1.2.1 Методы Data Mining

Рассмотрим основные методы Data Mining[8]:

а) **Статистические методы**

К базовым методам Data Mining традиционно причисляют все подходы, использующие элементы теории статистики. Последние версии почти всех известных статистических пакетов включают наряду с традиционными статистическими методами также элементы Data Mining. В качестве примеров наиболее мощных и распространенных статистических пакетов можно назвать SAS (компания SAS Institute), SPSS (компания SPSS), STATGRAPICS (компания Manugistics), STATISTICA, STADIA и другие.

б) **Полный и ограниченный перебор**

К базовым методам Data Mining принято относить также алгоритмы, основанные на переборе. Простой перебор всех исследуемых объектов требует $O(2^N)$ операций, где N – количество объектов. Следовательно, с увеличением количества данных объем вычислений растет экспоненциально, что при большом объеме делает решение любой задачи таким методом практически

невозможным. Наиболее ярким современным представителем реализации этого подхода является система WizWhy компании WizSoft.

в) **Нечеткая логика**

основным способом исследования задач анализа данных является их отображение на формализованный язык и последующий анализ полученной модели. Неопределенность по объему отсутствующей информации у системного аналитика можно разделить на три большие группы:

— неизвестность;
— неполнота;
— недостоверность.

Недостоверность бывает физической (источником ее является внешняя среда) и лингвистической (возникает в результате словесного обобщения и обусловливается необходимостью описания бесконечного числа ситуаций ограниченным числом слов за ограниченное время). основной сферой применения нечёткой логики и во многом остается управление.

г) **Генетические алгоритмы**

Генетические алгоритмы относятся к числу универсальных методов оптимизации, позволяющих решать задачи различных типов (комбинаторные, общие задачи с ограничениями и без ограничений) и различной степени сложности. одним из наиболее востребованных приложений генетического алгоритма в области Data Mining является поиск наиболее оптимальной модели (поиск алгоритма, соответствующего специфике конкретной области). Эти алгоритмы удобны тем, что их легко распараллеливать.

д) **Нейронные сети**

Нейронные сети – это класс моделей, основанных на биологической аналогии с мозгом человека и предназначенных для решения разнообразных задач анализа данных после прохождения этапа, так называемого обучения на имеющихся данных. При применении этого метода, прежде всего, встает вопрос выбора конкретной архитектуры сети (числа "слоев"и количества "нейронов"в каждом из них). размер и структура сети должны соответствовать существу исследуемого явления.

е) **Деревья решений**

Деревья решения являются одним из наиболее популярных подходов к решению задачи классификации. они создают иерархическую структуру классифицирующих правил типа "если-то"(if-then), имеющую вид дерева. Для принятия решения, к какому классу отнести некоторый объект или ситуацию, требуется ответить на вопросы, стоящие в узлах этого дерева,

начиная с его корня. Большинство систем, построенных на основе технологии Data Mining, используют именно этот метод.

1.2.2 Фракталы в Data Mining

Фракталы особенно показали свою эффективность в области data mining. Для обработки данных в работе использовались следующие фрактальные техники: кластеризация и понижение размерности.

При кластеризации происходит разбиение элементов множества на группы в зависимости от их схожести. Самоподобие фракталов позволяет очень естественно определить кластеры, не ораничивая их какой-то конкретной формой. При фрактальной кластеризации кластеры образуются, основываясь на фрактальной размерности: внутри кластера точки имеют большую степень самоподобия, чем вне его. При добавлении новых точек вычисляется новая фрактальная размерность и происходит перерасчет. Уже образованные кластеры могут разбиваться на новые или соединяться в один.

При понижении размерности происходит исключение коррелирующих атрибутов отношения. атрибуты, которые могут быть получены на основе других, исключаются из дальнейшей обработки. Для выявления повторяющихся данных используется свойство фрактальной размерности, оценка степени свободы набора данных.

1.3 Базы данных

База данных (БД) - некоторая совокупность данных, организованная в соответствии с определёнными правилами и имеющая определённую структуру, редактируемая при помощи специальной системы управления базами данных (СУБД).

По типу хранимой информации БД делятся на:

а) документальные - единицей хранения является документ, основная часть которого — неструктурированный текст, результатом выборки является не конкертная информация, а список документов, в определенной мере содержащих искомую информацию;

б) фактографические - единицей хранения является факт (элемент содержательной информации), часто используются в справочных целях;

в) лексикографические - единицей хранения является лексическая единица, часто представляют собой машиночитаемые словарные массивы.

По характеру организации хранения данных и обращения БД делятся на:

а) персональные или локальные - предназначены для локального использования одним пользователем;

б) централизованные - предполагают свое использование многими пользователями;

в) распределенные - помимо многопользовательского доступа, могут физически храниться на разных носителях, оставаясь при этом логически одним целым.

По типу использованной модели структурированные БД делятся на:

а) иерархические - структура такой базы представляет собой дерево, все объекты хранятся в виде определеной сущности;

б) сетевые - структура такой базы представляет собой сеть, то есть каждый элемент может иметь несколько предков;

в) реляционные - основаны на реляционной модели;

г) мультимодельные - поддерживают несколько моделей баз данных.

Большинство баз данных являются строчными, но в этой работе особый интерес имеет другой, координально противоположный тип архитектуры - базы данных с колоночным хранением, так как метод фрактального поиска работает преимущственно со столбцами таблиц баз данных.

Колоночные СУБД появились и развивались параллельно со строчными.

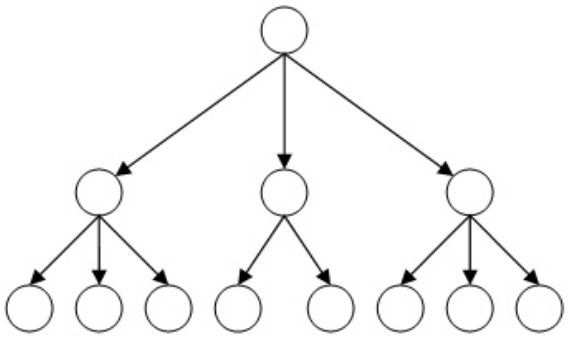

Рисунок 1.11 — Структура иерархической БД

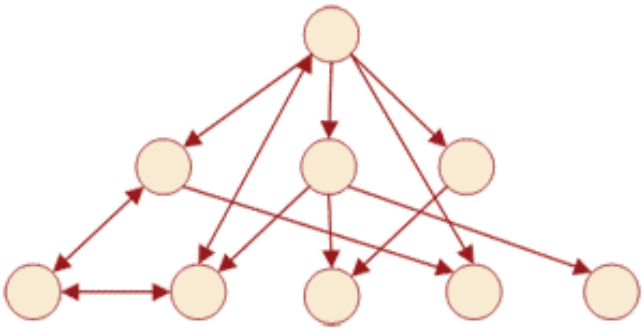

Рисунок 1.12 — Структура сетевой БД

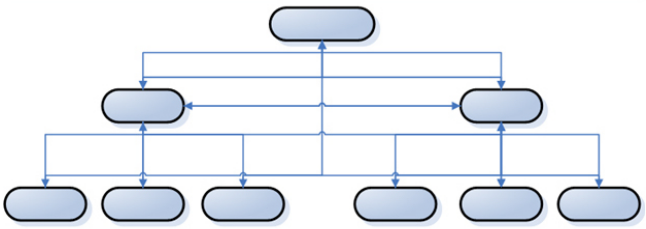

Рисунок 1.13 — Структура реляционной БД

Первой коммерческой колоночной СУБД считается TAXIR (1969), ориентированная на хранение и поиск биологических данных[11].
Колоночные БД обладают важным для этой работы свойством: они способны быстро выполнять сложные запросы с произвольными столбцами таблицы.

1.3.1 Реляционная модель

Реляционная модель данных - созданная Эдгаром Коддом логическая модель данных, описывающая:

а) структуры данных в виде (изменяющихся во времени) наборов отношений;

б) теоретико-множественные операции над данными: объединение, пересечение разность и декартово произведение;

в) специальные реляционные операции: селекция, проекция, соединение и деление;

г) специальные правила, обеспечивающие целостность данных.

В сравнении с иерархической и сетевой моделью данных, реляционная модель отличается более высоким уровнем абстракции данных. Реляционная модель является удобной и наиболее привычной формой представления данных, так в настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные коммерческие СУБД.

Реляционная модель данных предусматривает структуру данных, обязательными объектами которой являются:

а) отношение - плоская (двумерная) таблица, состоящая из столбцов и строк;

б) атрибут - поименованный столбец отношения;

в) домен - набор допустимых значений для одного или нескольких атрибутов;

г) кортеж - строка отношения;

д) степень - количество атрибутов;

е) кардинальность - количество кортежей, которое содержит отношение;

ж) первичный ключ - уникальный идентификатор для таблицы.

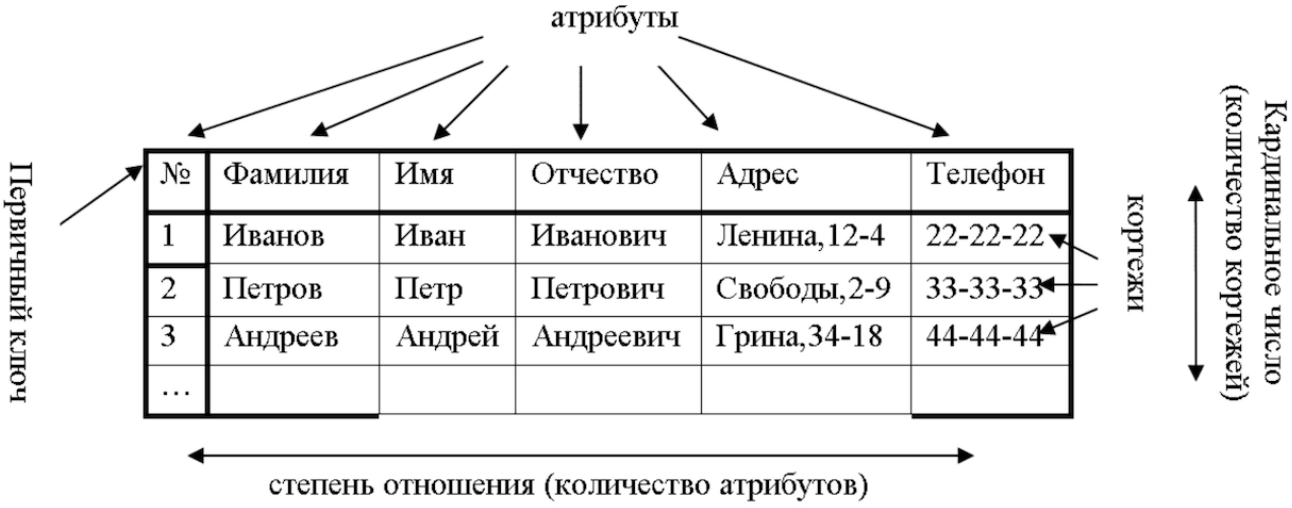

Рисунок 1.14 — Схема реляционной базы данных

Записи в таблице хранятся упорядоченными по ключу. Ключ может быть простым, состоящим из одного поля, и сложным, состоящим из нескольких полей. Сложный ключ выбирается в тех случаях, когда ни одно

поле таблицы однозначно не определяет запись.

Кроме первичного ключа в таблице могут быть вторичные ключи, называемые еще внешними ключами. Внешний ключ – это поле или совокупность полей, чьи значения имеются в нескольких таблицах и которое является первичным ключом в одной из них. Значения внешнего ключа могут повторяться в некоторой таблице. Внешний ключ обеспечивает логическую последовательность записей в таблице, а также прямой доступ к записи.

По первичному ключу всегда отыскивается только одна строка, а по вторичному – может отыскиваться группа строк с одинаковыми значениями первичного ключа. Ключи нужны для однозначной идентификации и упорядочения записей таблицы, а индексы для упорядочения и ускорения поиска.

Индекс - объект базы данных создаваемый с целью повышения производительности поиска данных, представляет собой сбалансированное дерево поиска. Повышая скорость поиска по таблице, он замедляет скорость вставки в нее[12].

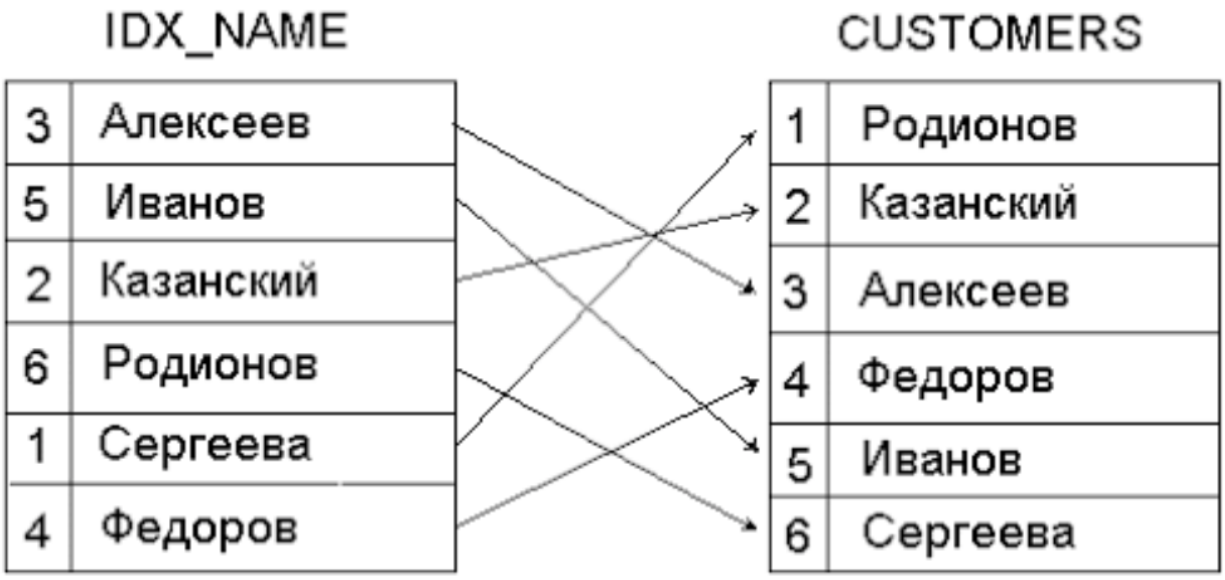

Рисунок 1.15 — Пример индекса

1.3.2 Parquet

Parquet является бинарным колоночно-ориентированным форматом данных, считается идеальным форматом для big data. Этот формат использует архитектуру, основанную на "уровнях определения"и "уровнях повторения что позволяет довольно эффективно кодировать данные, а информация о схеме выносится в отдельные метаданные[4]

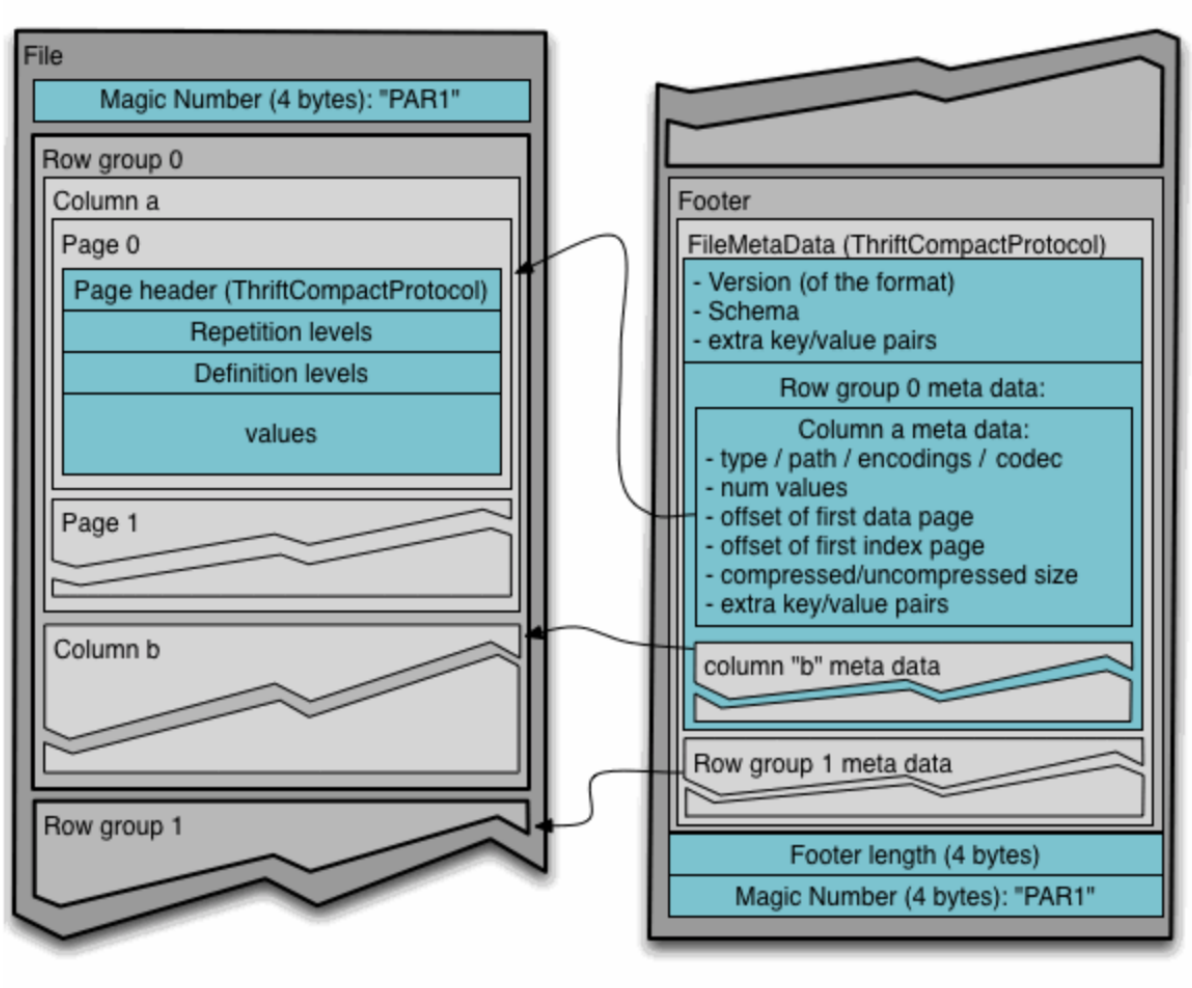

Рисунок 1.16 — Структура Parquet

Формат parquet обладает следующими преимуществами:

а) колоночная структура позволяет эффективно сжимать данные;
б) запросы не считывают лишние данные;
в) parquet хранится по сути в обычных файлах, что упрощает процедуры резервного копирования и репликации.

Формат parquet обладает следующими недостатками:

а) не поддерживает транзакции;
б) не поддерживает изменения структуры таблицы

в) не поддерживает изменение типа хранимых данных;

1.4 Алгоритмы сжатия баз данных

Существует множество алгоритмов сжатия баз данных, но все они, так или иначе, основаны на одном из двух способов уменьшения избыточности: изменение содержания данных или изменение их структуры. Некоторые алгоритмы совмещают два эти способа
Ниже будут изложены существующие алгоритмы сжатия баз данных.

1.4.1 Легкое сжатие

Алгоритмы легкого сжатия обычно основанны на словарном кодировании, то есть на замене длинных значений более короткими.

Примеры алгоритмов легкого сжатия:

а) **Подавление одинаковых значений**
Работает со столбцами, в которых много повторяющихся ненулевых значений. Создается словарь, в него помещются значения столбца, в исходной таблице эти значения замещаются позцией из словаря

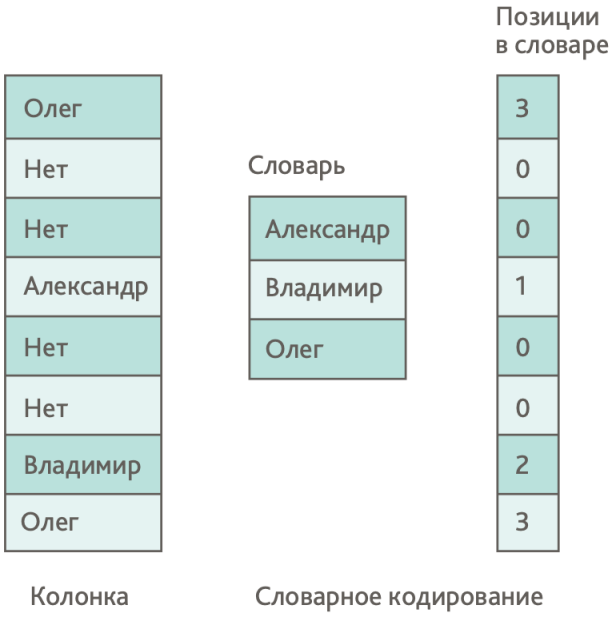

Рисунок 1.17 — Подавление одинаковых значений

б) **Кодирование по длинам серий**
Словарь хранит значение, длину серии и индекс начала серии. особенно хорошо этот алгоритм работает в отсортированных столбцах.

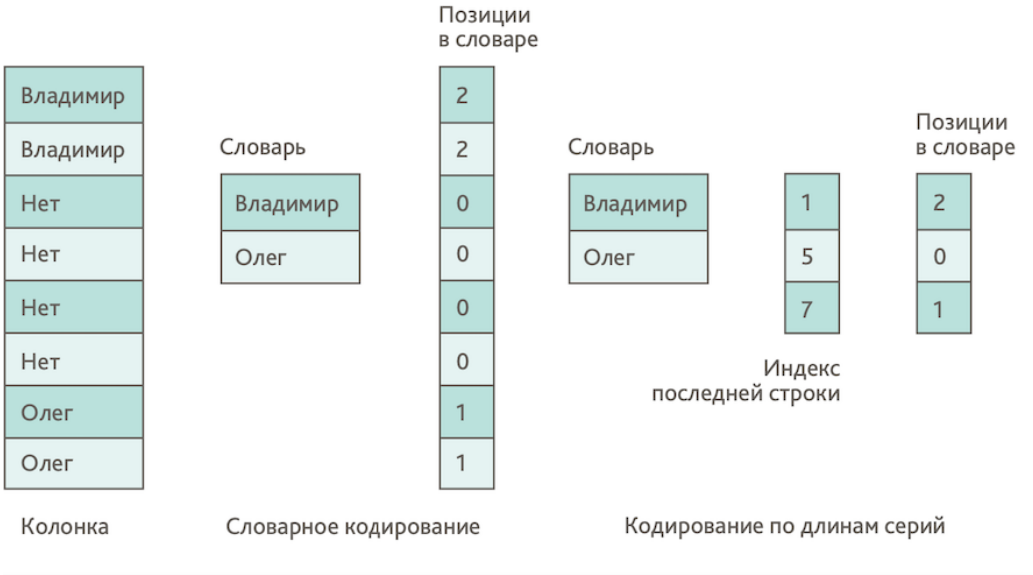

Рисунок 1.18 — Кодирование по длинам серий

в) **Непрямое кодирование**

Делит столбец на блоки одинакового размера, затем сжимает часть их них, создавая для кажого сжимаемого блока свой словарь

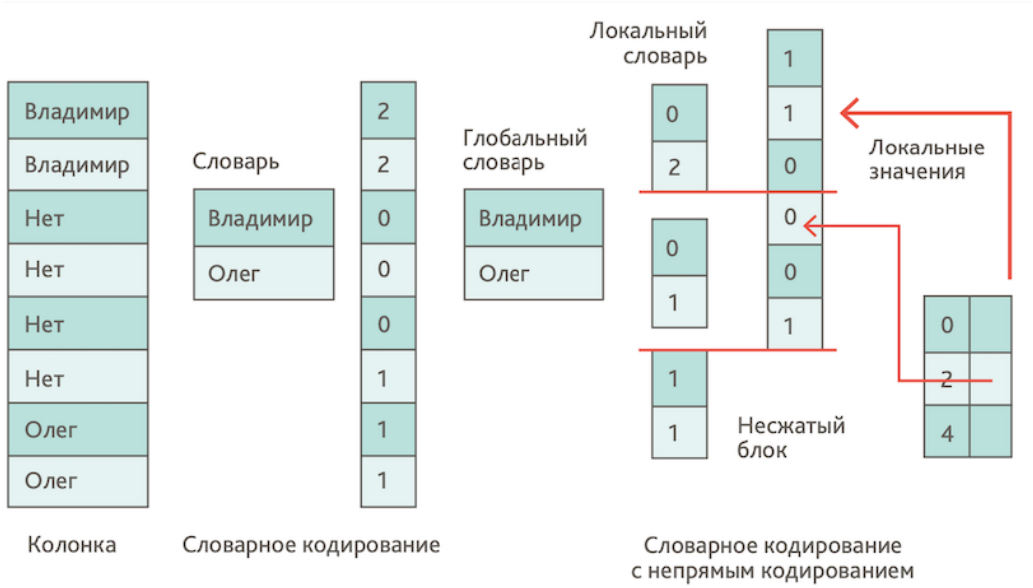

Рисунок 1.19 — Непрямое кодирование

г) **Битовое сжатие**

Вместо того чтобы хранить каждое значение с помощью встроенного целочисленного типа данных, битовое сжатие использует только необходимое количество битов для представления значений. То есть значения разбиваются на части фиксированной ширины, причем размер части определяется по величине наибольшего значения, которое нужно закодировать.

д) **Байтовое кодирование**

По сравнению с битовым сжатием байтовое кодирование использует байты, а не биты как основную единицу для создания частей памяти. Значения разделяются на части по 7 бит, и каждая часть хранится в байте. оставшийся ведущий бит используется для определения, конец ли это значения или нет. Таким образом, каждое значение может занимать разное количество

1.4.2 Метод Хаффмана

Метод Хаффмана как один из основных и наиболее популярных способов экономного кодирования неоднократно рассматривался применительно к задаче сжатия в СУДБ.
Кодирование Хаффмана является простым алгоритмом для построения кодов переменной длины, имеющих минимальную среднюю длину.

Метод начинается составлением списка символов алфавита в порядке убывания их вероятностей. Затем от корня строится дерево, листьями которого служат эти символы. Это делается по шагам, причем на каждом шаге выбираются два символа с наименьшими вероятностями, добавляются наверх частичного дерева, удаляются из списка и заменяются вспомогательным символом, представляющим эти два символа. Вспомогательному символу приписывается вероятность, равная сумме вероятностей, выбранных на этом шаге символов. Когда список сокращается до одного вспомогательного символа, представляющего весь алфавит, дерево объявляется построенным. Завершается алгоритм спуском по дереву и построением кодов всех символов.

В чистом виде метод Хаффмана не может конкурировать со словарными схемами по степени сжатия, что предопределяет его нишу вспомогательной техники. Достоинство подхода в простоте и скорости декодирования. Метод допускает простую аппаратную реализацию[7].

1.4.3 Методы Зива-Лемпела

одной из самых популярных тяжелых схем сжатия является Lempel-Ziv кодирование. Словарные методы, относимые к этому классу, можно разделить на два семейства: LZ77 и LZ78.

В методах семейства LZ77 роль словаря играет порция уже обработанных данных. Последовательность обычно кодируется указанием начала эквивалентной фразы в словаре (смещения) и длины совпадения.

Пара <смещение, длина совпадения> в этом случае является указателем. если кодируемый элемент отсутствует в словаре, то он некоторым образом помечается и представляется как есть. Такой элемент называется литералом. Из способа формирования словаря следует, что методы семейства LZ77 являются адаптивными.

На практике схемы типа LZ77 используются совместно с алгоритмами статистического кодирования указателей и литералов.

В схемах семейства LZ78 в словарь включаются не все последовательности, встреченные в обработанном массиве данных, а лишь "перспективные"с точки зрения появления в дальнейшем. В отличии от семейства LZ77, в словаре не может быть одинаковых фраз. Указатели фраз кодируются словами фиксированной длинны, определяемой размером словаря. В рамках семейства LZ78 несложно реализовать эффективное неадаптивное и полуадаптивное сжатие, при котором словарь строится заранее[7].

2 Конструкторский раздел

В данном разделе описаны основные объекты и алгоритмы, используемые в системе.

2.1 Алгоритм фрактального поиска

Для решения задачи фрактального поиска воспользуемся теорией реляционных баз данных. Ввёдем основные термины.

Домен – некоторое конечное множество данных, обозначается как D_i , где i – номер домена. Отдельный элемент домена обознается как d_i, где i – также номер домена.

Полное декартово произведение множеств – набор всевозможных сочетаний из n элементов каждое, где каждый элемент берётся из своего домена. Описание: $D_1 \times D_2 \times \cdots \times D_n$.

Отношение R – подмножество декартова произведения множеств D_1, $D_2 \ldots D_n$, необязательно различных. Описание: R $\subseteq D_1 \times D_2 \times \cdots \times D_n$. Число n называется степенью отношения.

Атрибутом называют домен, входящий в отношение. Степень отношения определяет количество атрибутов в отношении.

Схемой отношения S называется перечень имён атрибутов данного отношения с указанием домена, к которому они относятся. Описание: $S_R = (A_1, A_2, \ldots A_n.)$, $A_i \subseteq D_i$.

Каждому имени атрибута A_i ставится в соответствие множество D_i — множество значений атрибута A_i, 1⩽i⩽n, D — объединение D_i, 1⩽i⩽n.

Отношение R со схемой S — это конечное множество отображений t_1, t_2,..., t_n из S в D; причем каждое отображение t должно удовлетворять следующему ограничению: t(A_i) принадлежит D_i, 1⩽i⩽n. Эти отображения называются кортежами.

Из последнего определения видно, что отношение состоит из нескольких частей, кортежей. Именно кортеж несёт в себе информацию о структуре отношения.

Алгоритм фрактального поиска рассматривает именно кортежи. Мы рассматриваем атрибуты кортежа как отдельные части, сохраняя при этом сведения о структуре в целом.

Так как каждый атрибут имеет конечное множество значений, множество значений набора атрибутов будет также конечно. В этом случае можно предположить, что домены в отношении могут иметь идентичную структуру и одинаковые значения.

Принимая это предположение, возможно выделить домены в отношении, определить взаимосвязь атрибутов и фрактально сжать таблицу, сохранив при этом её структуру. Фрактальное сжатие поможет не только более компактно хранить имеющиеся данные, но и обнаружить в данных нетривиальные корреляции, полезные для дальнейшего анализа данных.

Основная идея алгоритма фрактального поиска заключается в подборе такого множества доменов, которое не только полностью опишет структуру отношения, но и позволит представить данные в базе минимальным количеством столбцов[1].

На рисунке 2.1 представлена структура фрактального поиска:

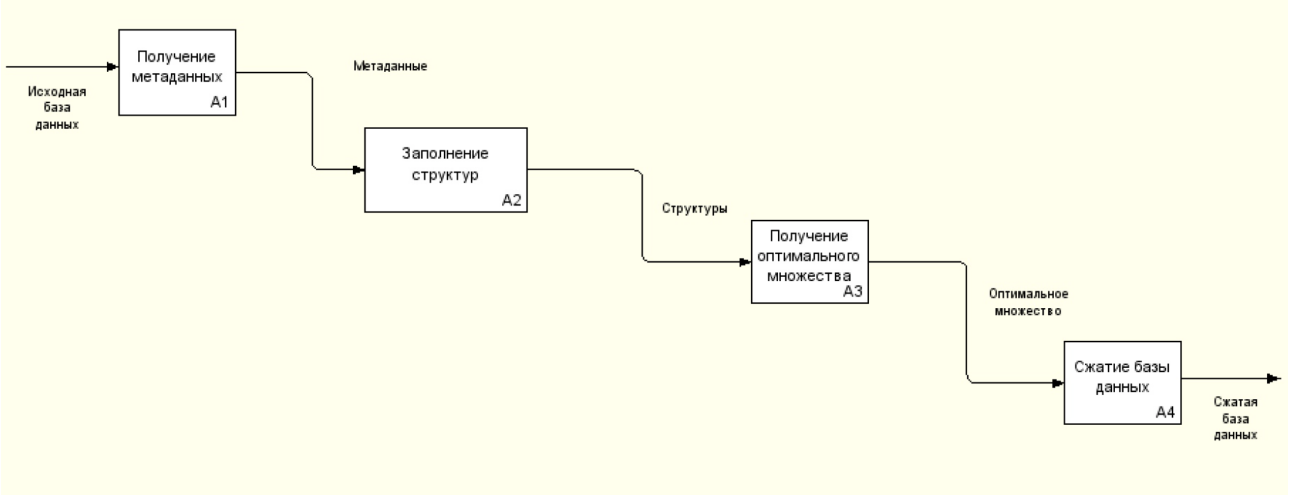

Рисунок 2.1 — Диаграмма фрактального поиска

2.2 Работа с доменами

Пусть в нашей таблице имеется n атрибутов и в каждый домен входит k атрибутов, 1⩽k⩽n.

Количество различных доменов из k атрибутов: $C_n^k = \frac{(n!)}{(k!(n-k)!)}$

Количество всех возможных различных доменов: $\sum_{k=1}^{n} C_n^k = \sum_{k=1}^{n} \frac{(n!)}{(k!(n-k)!)}$

Ниже на рисунке 2.2 показан график роста возможных доменов от количества аттрибутов в таблице:

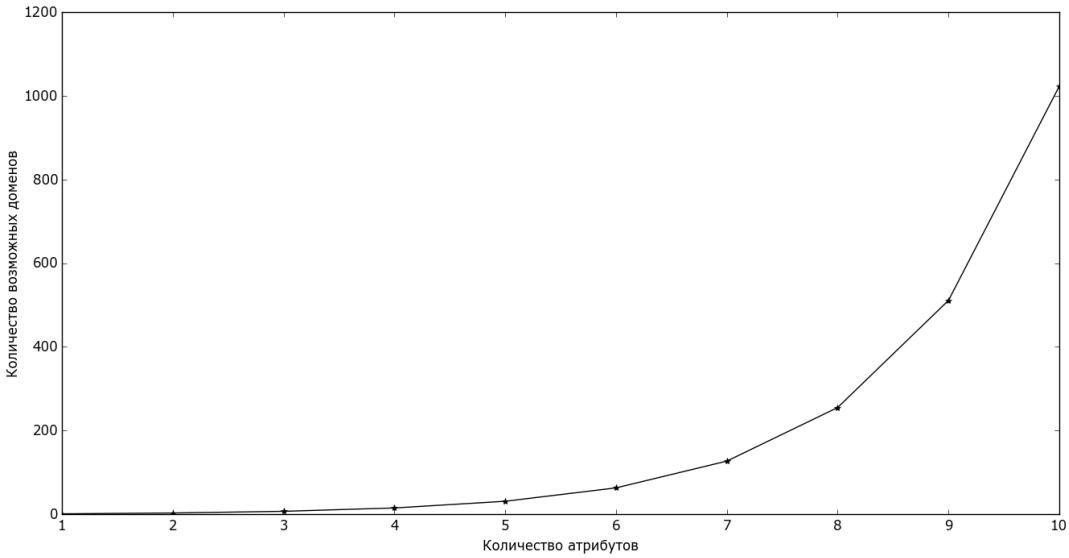

Рисунок 2.2 — Рост количества возможных доменов

Как видно из графика, с ростом количества атрибутов резко возрастает количество всех возможных доменов. Как правило, при формировании отношения число входящих в него атрибутов стараются ограничивать 15, что дает нам количество возможных доменов не более 32767, однако при проектировании схемы базы данных могут и не придерживаться этого правила. Сложность во многом будет зависеть именно от количества атрибутов в таблице.

Одной из важнейших характеристик домена является количество различных значений в нём. Чем меньше различных значений в домене, тем больше информации о структуре базы данных он отображает. Количество различных значений домена можно определить как произведение количества различных значений атрибутов, входяших в него, но на практике это

число оказывается меньше. Поиск количества различных значений домена является самой затратной и самой неоптимизируемой частью алгоритма фрактального поиска, поэтому имеет смысл ограничить размер доменов, чтобы сократить время выполнения данной части.

После того, как будет получено количество различных значений для каждого из множества доменов, необходимо получить оптимальное множество доменов. Оптимальное множество доменов полностью описывает таблицу и определяется как $\mathrm{D} = D_1 + \cdots + D_m$.

Оптимальное множество доменов должно удовлетворять следующим требованиям:

а) все атрибуты должны быть в единственном экземпляре;

б) сумма количества различных значений доменов должна быть минимальна.

Алгоритм поиска оптимального множества является рекурсивной функцией: последовательно добавляются новые домены до тех пор, пока не будут просмотрены все атрибуты или пока количество различных значений доменов не превышает текущего минимума. При первом запуске минимум определяется как произведение количества атрибутов на количество строк, т.е. каждая ячейка считается уникальным значением. Если в базе данных существует несколько оптимальных множеств, данная реализация алгоритма выбирает первое из них.

2.3 Сохранение данных в СУБД

Метод сохраняет в базе следующие таблицы:

а) сжатую таблицу;
б) таблицу метаданных;
в) словари доменов;

Таблица с метаданными сохраняет информацию о столбцах исходной таблице и их связях со столбцами в сжатой таблице. Пример такой таблицы показан на рисунке 2.3:

id	real_index	column_type_name	column_name	packed	domen_name
1	0	Integer	ID	0	ID
2	1	Varchar(35)	Name	0	Name
3	2	Varchar(11)	Sex	0	Sex
4	3	Varchar(12)	Age	1	domen1
4	11	Varchar(16)	Medal	1	domen1
5	4	Varchar(35)	Team	1	domen2
5	5	Varchar(13)	NOC	1	domen2
6	6	Integer	Year	1	domen3
6	8	Varchar(32)	City	1	domen3
7	7	Varchar(16)	Season	0	Season
8	9	Varchar(35)	Sport	1	domen4
8	10	Varchar(70)	Event	1	domen4

Рисунок 2.3 — Пример таблицы метаданных

Сжатая таблица и словари доменов должны загружаться в базу данных из csv-файлов.

2.4 Key-value хранилище

Данный модуль решает следующие задачи:

а) получение всего хранилища из базы;
б) сохрание всего хранилища в базу;

Для обоих этих операций используется таблица метаданных: в ней есть поля, из которых вычисляется количество и имена словарей доменов. То есть, располагая информацией из этой таблицы, мы можем узнать какие словари нужно сохранять или получить из базы.

Чтобы получить информацию из словарей метод производит выборку для каждого домена, затем загружает ее в структуру из вложенных словарей.

Так как некоторые операции, такие как изменение строк, производятся только с хранилищем, необходим механизм его сохранения обратно в базу. Чтобы выполнить обратную операцию сохранения хранилище обратно в базу данных, необходимо для каждого домена создать временный csv-файл, затем заполнить на его основе таблицу со словарем соответствующего домена.

Также хранилище содержит информацию о названиях колонок и их порядок в исходной таблице для реализации преобразования sql-запросов. На рисунке 2.4 показана структура такого хранилища:

Рисунок 2.4 — Структура хранилища

2.5 Модуль преобразования sql-запросов

В сжатой базе данные хранятся в нечитаемом для людей виде, а при работе с ней конечный пользователь предпочтет получить данные в формате, с которым можно работать, в связи с чем необходим механизм проеобразования запросов.

Данный модуль имеет возможность преобразовывать запросы к исходной базе в запросы к сжатой базе путем подмены значений и столбцов, если они входят в сжатый домен, так же модуль способен делать обратное преобразоние сжатых данных в читаемый вид.

Для всех этих операций, чтобы не делать большое количество дополнительных запросов в базу, требуется дополнительное key-value хранилище, которое значительно облегчает процессы шифрования и дешифрования данных.

Из тех же соображений пресечения лишних запросов, фильтрация результатов некоторых комманд SELECT переложена с базы данных на данный модуль. отчего модуль реализовывает неполный функционал SQL, к примеру, не осуществляет комманды JOIN и ограничен в логических предикатах.

2.5.1 Алгоритм преобразования запросов select

Алгоритм получает на вход sql-запрос, из него получает колонки для запроса, а также логические предикаты, сортировку и ограничение на количество выходных строк, если такие есть. Далее алгоритм преобразовывает колонки запроса и значения логических предикатов к колонкам сжатой таблицы и выполняет запрос. Далее алгоритм преобразовывает данные обратно, и если столбец сортировки был сжат, то производится сортировка по этому столбцу. На рисунке 2.5 представлена схема данного алгоритма:

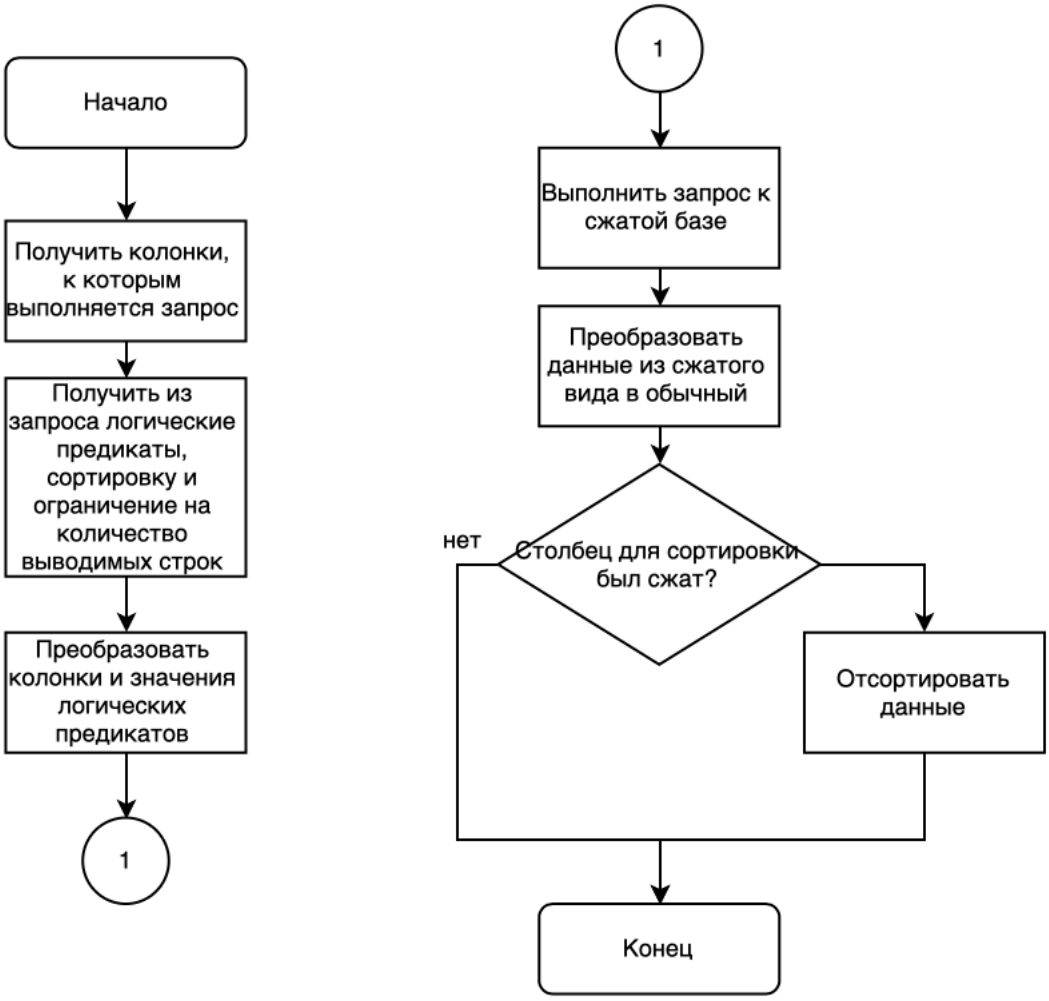

Рисунок 2.5 — Схема алгоритма преобразования запросов select

2.5.2 Алгоритм преобразования запросов delete

Алгоритм преобразования запросов delete получает на вход sql-запрос, из него получает колонки для запроса и логические предикаты. Далее, если колонки для запроса не сжаты, то алгоритм просто выполняет исходный запрос. В противном случае он находит все ключи в словаре домена, значение которых удовлетворяют предикатам, а затем выполняет отдельный запрос для каждого ключа. Схема этого алгоритма показана на рисунке 2.6:

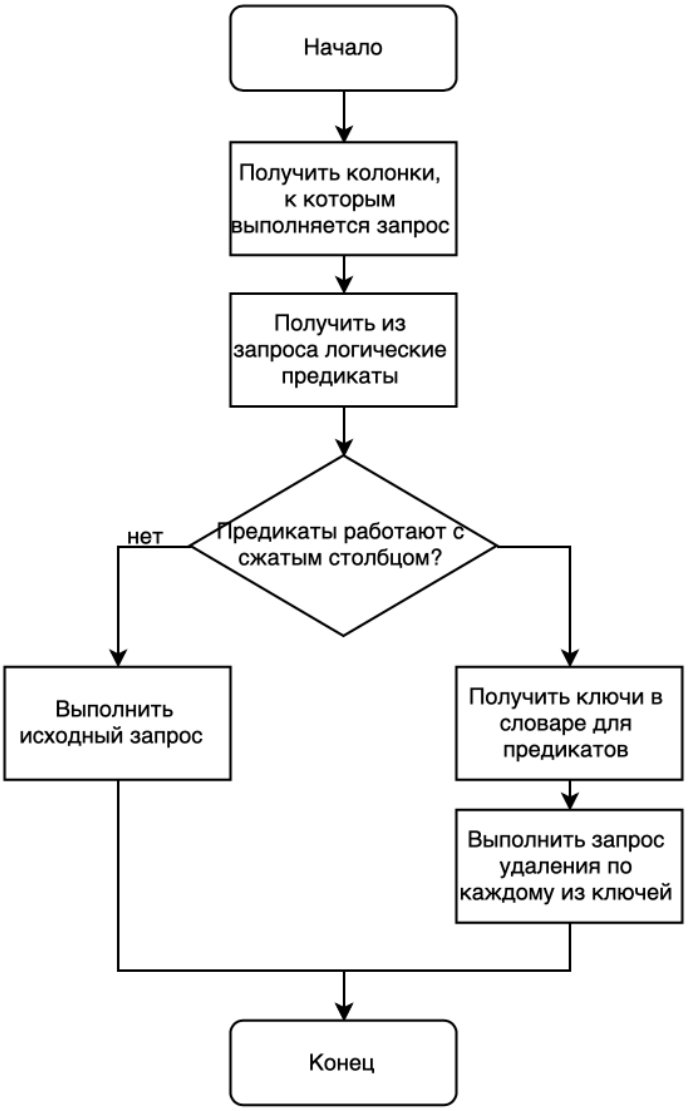

Рисунок 2.6 — Схема алгоритма преобразования запросов delete

2.5.3 Алгоритм преобразования запросов update

Алгоритм преобразования запросов update получает на вход sql-запрос, из него получает колонки для запроса и логические предикаты. Далее, если колонки для запроса не сжаты, то алгоритм просто выполняет исходный запрос. В противном случае он находит все ключи в словаре домена, значение которых удовлетворяют предикатам, а затем производится замена значений в словаре. На рисунке 2.7 находится схема данного алгоритма:

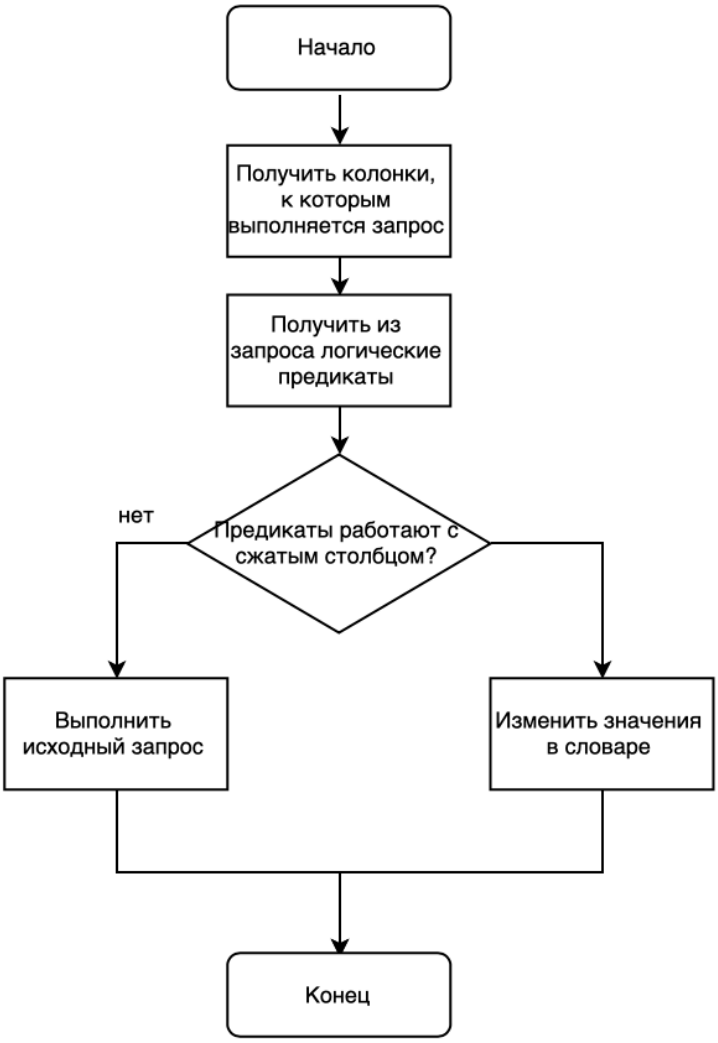

Рисунок 2.7 — Схема алгоритма преобразования запросов update

2.5.4 Алгоритм преобразования запросов insert

Алгоритм преобразования запросов insert получает на вход sql-запрос, получает из него значения атрибутов, сжатых в домены, и затем для каждого домена проверяет, есть ли такое значение в словаре. Если он не находит такого, то он добавляет новое значение в словарь. Затем значения этих атрибутов меняются на позицию значения из словаря и выполняется запрос. Схема алгоритма преобразования запросов insert показана на рисунке 2.8.

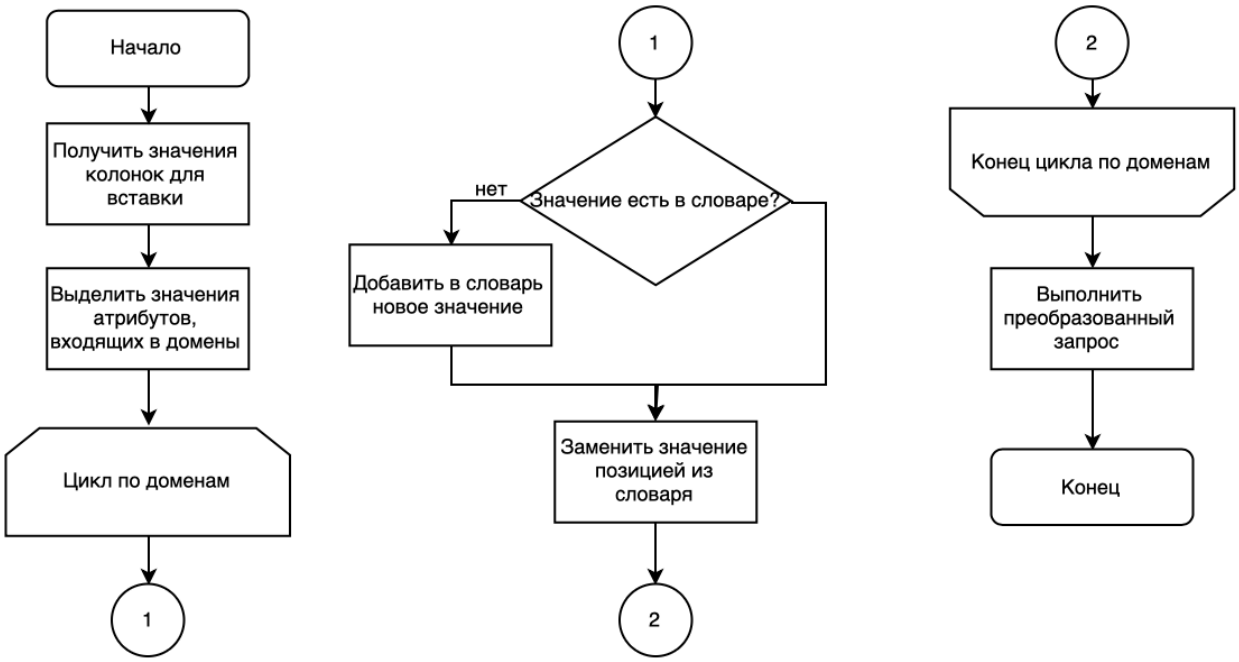

Рисунок 2.8 — Схема алгоритма преобразования запросов insert

3 Технологический раздел

3.1 Выбор средств разработки

3.1.1 Выбор языка программирования

В качестве языка программирования выбран язык Python. Он обладает одновременно широким функционалом использования и сравнительной простотой разработки. Python является высокоуровневым языком программирования, поддерживает объектно-ориентированную парадигму программирования. Язык имеет внушительное количество библиотек, в том числе и необходимые для данной работы.

3.1.2 Выбор среды разработки

Средой разработки была выбрана IDE PyCharm Community Editon компании JetBrains, специализирующейся на производстве инструментов для профессиональной разработки программного обеспечения. PyCharm CE распространяется на условиях лицензии Apache 2.0, то есть разрешает свободное использование и модификацию исходного кода.

Данная среда выбрана из-за следующих удобств и преимуществ:

а) наличие статического анализатора;

б) удобный менеджер по работе с виртуальными окружениями и библиотеками;

в) интеграция с системой контроля версий (VCS) Git;

г) возможность создания и настройка нескольких конфигураций запуска.

3.1.3 Выбор используемых библиотек

Для данной работы главной используемой библиотекой служит библиотека vertica-python. По своей сути это нативный клиент для базы данных Vertica от её создателей. Так же в системе используется библиотека ast (Abstract Syntax Trees), для вычислений строк как программного кода на языке Python. Для разработки пользовательского графического интерфейса исполь

3.2 Выбор базы данных

В качестве базы данных для реализации метода фрактального поиска была выбрана колоночная СУБД "Vertica". База данных распространяет по условиям лицензии, позволяеющей бесплатный доступ с ограничением по объему занимаемой памяти в 1 Tb. При разработке использовалась версия

Vertica Community Edition 9.3.1, предустановленная на виртуальную машину с операционной системой Centos 7.

База данных Vertica была выбрана ввиду следующих преимуществ:

а) колоночная структура, позволяющая ускорить выборку отдельных атрибутов таблицы;

б) наличие бесплатной лицензии;

в) встроенное сжатие, с которым можно сравнивать предложенный метод в ходе его реализации;

г) удобные средства по работе с базой данных в формате веб-сайта

3.3 Система контроля версий

Во время процесса разработки мною была использованна система контроля версий Git. Система контроля версий с помощью репозиториев решает проблемы с переносом программного кода на другие устройства, его резервным копированием, а так же дает возможность разделения версий продукта во время разработки, что позволяет при внесении изменений или модифакциях всегда иметь рабочию версию проекта.

Из всех систем была выбрана именно Git по следующим причинам:

а) удобная система ветвения;

б) интеграция с выбранной средой разработки;

в) наличие большого количества платформ, поддерживающих репозитории Git.

3.3.1 Требования к компьютеру

Разрабатываемая система имеет ряд требований к компьютеру пользователя, в том числе:

а) язык программирования Python, версия не ниже 3.6;

б) библиотека vertica-python;

в) база данных Vertica, версия не ниже 9.3.

Так же метод фрактального поиска работает с большим количеством данных, из чего следует ограничение на оперативную память - не меньше 2 GB.

3.4 Архитектура системы

Используемые в системе классы:

а) атрибуты (Attribute);
б) таблица (Table);
в) домен (Domen);
г) множество всех доменов (DomenSearch);
д) множество оптимальных доменов (OptimalDomen);
е) модуль преобразования sql-запросов (db_operations);
ж) key-value хранилище (dict_operations);

На рисунке 3.1 представленна диаграмма классов модуля сжатия метода фрактального поиска.

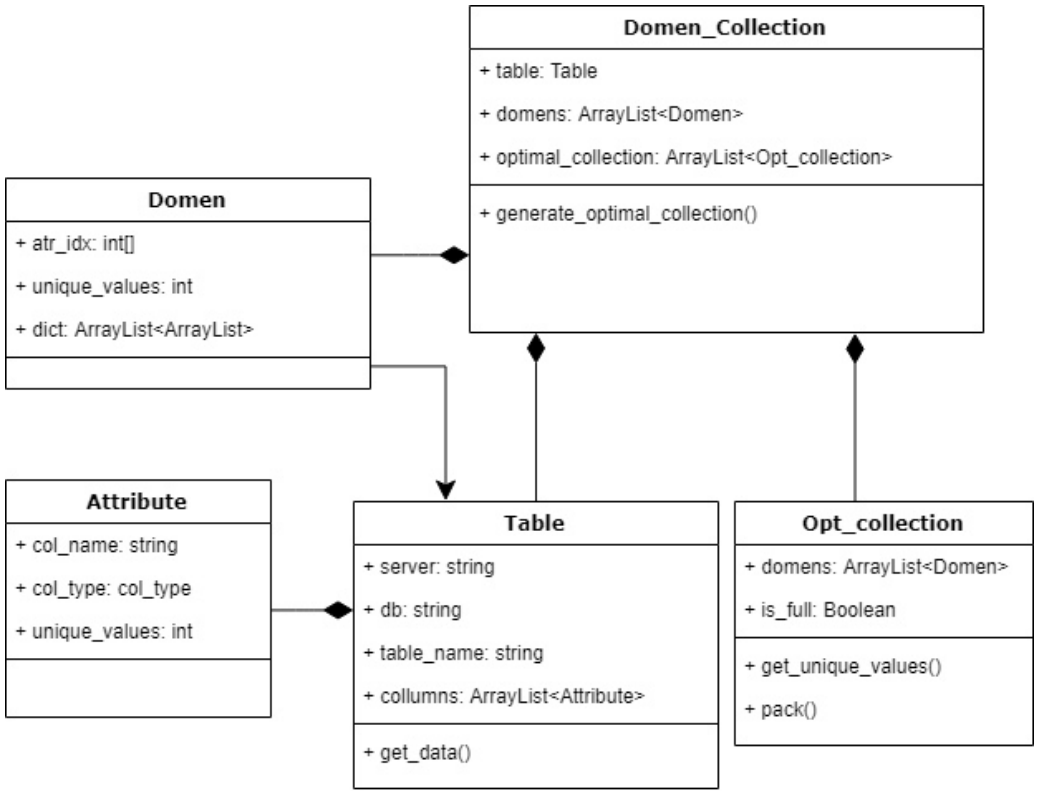

Рисунок 3.1 — Диаграмма классов модуля сжатия

Класс атрибута описывает столбец обрабатываемой таблицы. Класс содержит имя столбца, тип данных, который он содержит, а также количество уникальных значений. На рисунке 3.2 показан листинг класса аттрибута.

```
class Attribute(object):
    def __init__(self, col_name, col_type, unique_values):
        self.col_name = col_name
        self.col_type = col_type
        self.unique_values = unique_values

    def is_packable(self, rows):
        if self.unique_values * 2 > rows:
            return False
        return True
```

Рисунок 3.2 — Класс атрибута

Класс таблицы описывает обрабатываемую таблицу. Класс содержит имя столбца, её размер, а также лист атрибутов-столбцов. Помимо этого класс содержит методы получения данных. На рисунке 3.3 показан листинг класса таблицы.

```
import pyodbc
import bin_mod.methods as cfg
from bin_mod.attribute import Attribute


class Table(object):
    def __init__(self, table_name):
        self.server = cfg.server
        self.db = cfg.db
        self.table_name = table_name
        meta = cfg.get_cursor()
        meta.execute("SELECT  * FROM " + table_name)
        self.columns = [Attribute(x[0], x[1],
                        cfg.get_unique_values(table_name, [x[0]]))
                        for x in meta.description[1:]]
        self.length = len(meta.fetchall())

    def get_columns(self):
        return self.columns

    def get_data(self):
        cursor = cfg.get_cursor()
        cursor.execute("SELECT  * FROM " + self.table_name)
        return cursor.fetchall()
```

Рисунок 3.3 — Класс таблицы

Класс домена описывает наименьший основной фрактальный объект, домен. Класс содержит лист индексов входящих в него атрибутов-столбцов, а также количество уникальных значений в данном домене. Класс включает в себя метод получения количества уникальных значений. На рисунке 3.4 показан листинг класса домена.

```
1  from bin_mod.table import Table
2  import bin_mod.methods as cfg
3
4
5  class Domen(object):
6      def __init__(self, table, atr_idx):
7          self.atr_idx = atr_idx
8          columns = table.get_columns()
9          self.atr_list = [columns[x] for x in atr_idx]
10         self.unique_values = 0
11         self.dict = []
12
13     def get_atr_idx(self):
14         return self.atr_idx
15
16     def try_to_put(self, val):
17         if val not in self.dict:
18             self.dict.append(val)
19             self.unique_values += 1
```

Рисунок 3.4 — Класс домена

Класс множества всех доменов описывает алгоритмы работы с множеством доменов. Класс содержит лист индексов входящих в него атрибутов-столбцов, а также количество уникальных значений в данном домене. Класс включает в себя метод получения количества уникальных значений. На рисунке 3.5 показан листинг класса коллекции доменов.

```
class DomenCollection(object):
    def __init__(self, table, max_domen_length):
        self.table = table
        # all domens
        columns = table.get_columns()
        columns_count = len(columns)
        packable_domens = []
        unpackable_atr = []
        skip = []
        # get unpackable attributes
        for i in range(len(columns)):
            if not columns[i].is_packable(table.length):
                unpackable_atr.append(Domen(table, [i]))
                skip.append(i)
        # get all packable domens
        for length in range(1, min(max_domen_length+1,
                                   len(columns))):
            iterator = [x for x in range(0, length)]
            iterator = check_domen(iterator, length, skip)
            while iterator[0] <= columns_count - length:
                packable_domens.append(Domen(table,
                                             [x for x in iterator]))
                iterator = next_domen(iterator, columns_count,
                                      skip)
        self.domens = packable_domens + unpackable_atr
        # all full collections
        full_collections = get_full_collections(table,
                packable_domens, unpackable_atr, columns_count)
        fill_collections(table, full_collections)
        self.opt_collection = get_min_collection(full_collections)
```

Рисунок 3.5 — Класс коллекции доменов

3.5 Интерфейс пользователя

Для реализации метода фрактального поиска с использованием библиотеки PyQt был написан пользовательский интерфейс. На рисунке 3.6 представлен вид главной страницы пользовательского интерфейса.

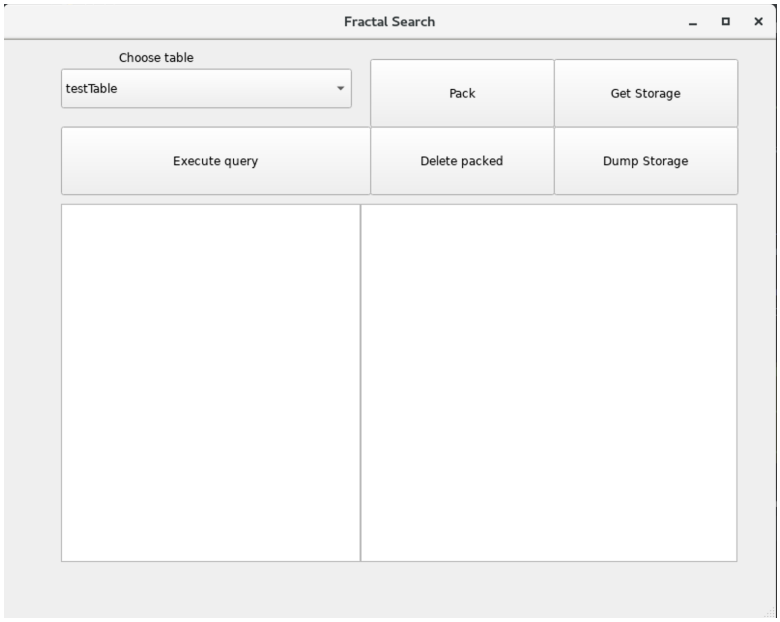

Рисунок 3.6 — Главная страница пользовательского интерфейса

На данном окне присутствует поле выбора таблицы, с которой метод будет работать, кнопки для выполнения сжатия таблицы, удаления сжатой таблицы и всех связанных с ней данных, кнопки для осуществления работы с хранилищем и выполнения sql-запросов. Ниже находятся два поля вывода: левое для логирования действий пользователя в приложении, второе для вывода результатов sql-запросов

Для операции сжатия таблицы или выполнения sql-запроса будет необходим дополнительный ввод, о чем система оповестит пользователя. Соответствующие всплывающие окна показаны на рисунках 3.7 и 3.8.

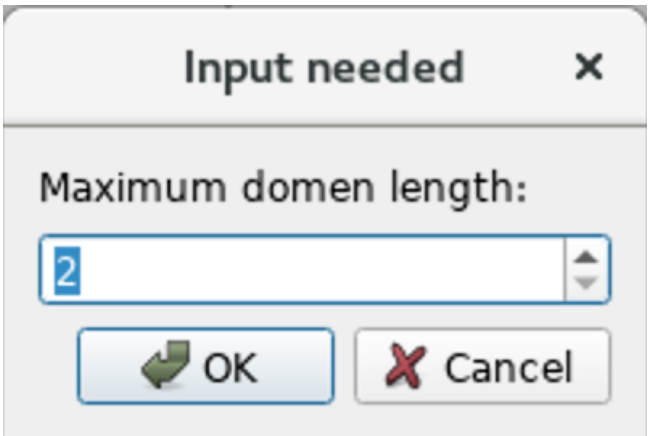

Рисунок 3.7 — Ввод максимальной длинны домена

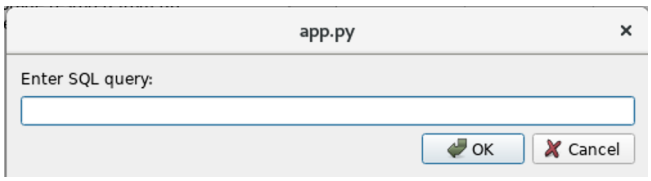

Рисунок 3.8 — Ввод sql-запроса

Так же в приложение оповещает пользователя об ошибках, произошедших во время выполнения операций. Пример такого сообщения показан на рисунке 3.9

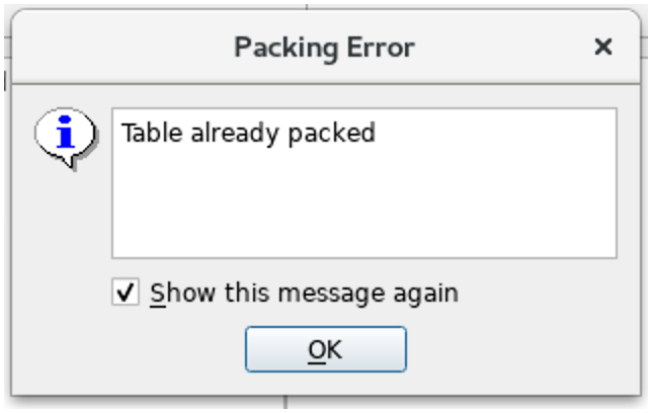

Рисунок 3.9 — Пример вывода сообщения об ошибке

4 Экспериментальный раздел

В данном разделе будут изложены результаты исследования метода фрактального поиска

4.1 База данных для исследования

Для проведения исследований были выбраны таблицы с контентом Netflix и историей олимпийских игр за 120 лет с параметрами, указанными в таблице 4.1:

Название	Количество строк	Количество столбцов	Количество столбцов, пригодных для сжатия
netflix_titles	6227	12	6
olympic	200000	12	8

Таблица 4.1 — Параметры таблиц

Данные таблицы были загружены в базу данных Vertica из файлов формата CSV, скачанных из [8].

4.2 Определение процента сжатия

При проведении данного эксперимента были исследовано влияние следующих факторов на процент сжатия:

а) количество максимальных доменов;
б) количество строк в таблице;
в) количество сжимаемых столбцов;

В таблице 4.2 представлены экспериментальные данные по фактическому количеству найденных доменов и полных коллекций в таблице olympic:

Максимальная длина домена	Количество найденых доменов	Количество найденных полных коллекций
2	57	9496
3	177	61136
4	387	99146

Таблица 4.2 — Рост сложности вычислений в таблице olympic

В таблице 4.3 представлены экспериментальные данные по фактическому количеству найденных доменов и полных коллекций в таблице netflix_titles:

Максимальная длина домена	Количество найденых доменов	Количество найденных полных коллекций
2	27	76
3	47	166
4	6	196

Таблица 4.3 — Рост сложности вычислений в таблице netflix_titles

Был проведен эксперимент с целью нахождения зависимости процента сжатия от длины таблицы и максимальной длины домена в таблице olympic. Результаты показаны в таблице 4.4

Максимальная длина домена	10000 строк	25000 строк	50000 строк
2	71.83	73.04	73.26
3	72.14	73.47	73.68
4	72.14	73.47	-

Таблица 4.4 — Зависимость процента сжатия от длины таблицы и максимальной длины домена в таблице olympic

Из таблицы видно, что коэффицент сжатия увеличивается одновременно с длиной таблицы и максимальной длиной домена. Также

видно, что для базы существует некоторый предел максимальной длины доменов, больше которого последующее сжатие бесполезно. Однако такого предела не наблюдается на вычисляемых значениях для длины таблицы

Был проведен аналогичный эксперимент для таблицы netflix_titles. Результат этого эксперимента представлен в таблице 4.5

Максимальная длина домена	Процент сжатия
2	1.96
3	1.96
4	1.96

Таблица 4.5 — Зависимость процента сжатия от максимальной длины домена в таблице netflix_titles

Таблица netflix_titles менее пригодна для сжатия из-за наличия в ней двух столбцов большой длины с уникальными значениями, которые занимают большую часть объема данных и не сжимаются, что тоже влияет на коэффициент сжатия. Сжимаемая часть настолько мала по сравнению с несжимаемой, что процент сжатия крайне мал и не меняется от длины домена

4.3 Изменения в структуре таблицы

В ходе проведения экспериментов с таблицей netflix_titles наблюдалось изменение структуры сжимаемых таблиц. На рисунке 4.1 показаны составы доменов сжатой таблицы netflix_titles с максимальной длинной домена 2. Учитывая длину таблицы (6227) можно установить факт

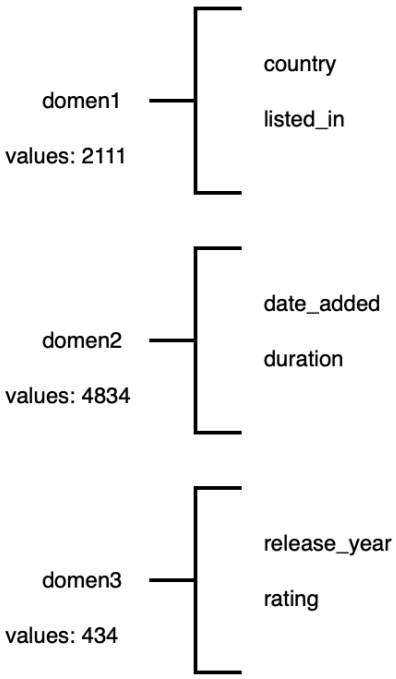

Рисунок 4.1 — Структура таблицы netflix_titles с максимальной длинной домена 2

корреляции во всех аттрибутах внутри данных доменов.

На рисунке 4.2 показаны составы доменов сжатой таблицы netflix_titles с максимальной длинной домена 3.

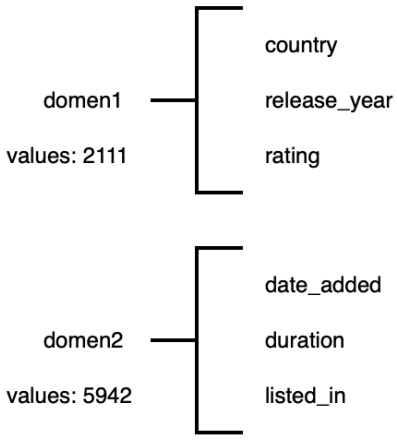

Рисунок 4.2 — Структура таблицы netflix_titles с максимальной длинной домена 3

На рисунке 4.3 показаны составы доменов сжатой таблицы netflix_titles с максимальной длинной домена 4.

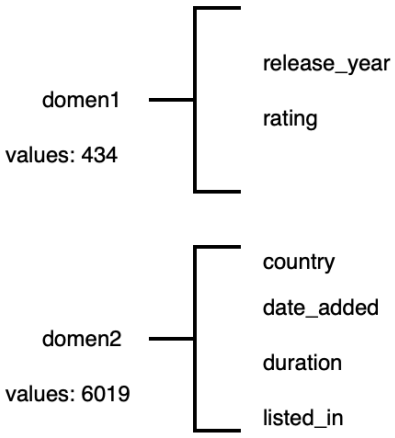

Рисунок 4.3 — Структура таблицы netflix_titles с максимальной длинной домена 4

С такими длиннами домена мы наблюдаем меньше корреляций внутри доменов, но зато и меньшее число аттрибутов вцелом. По таблице netflix_titles можно сделать вывод: при максимальной длинне домена, равной двум, домены отображают больше данных о структуре таблицы и имееют больше корреляций между своими аттрибутами.

Аналогичные изменения в структуре таблицы в ходе экспериментов претерпела и таблица olympic. На рисунках 4.4 и 4.5 показаны составы доменов сжатой таблицы olympic с максимальной длинной домена 2 и 3 соотвественно.

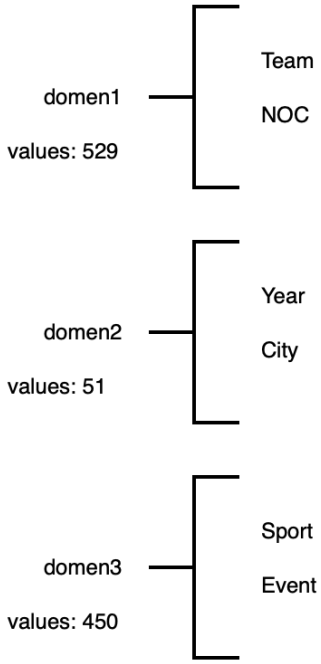

Рисунок 4.4 — Структура таблицы olympic с максимальной длинной домена 2

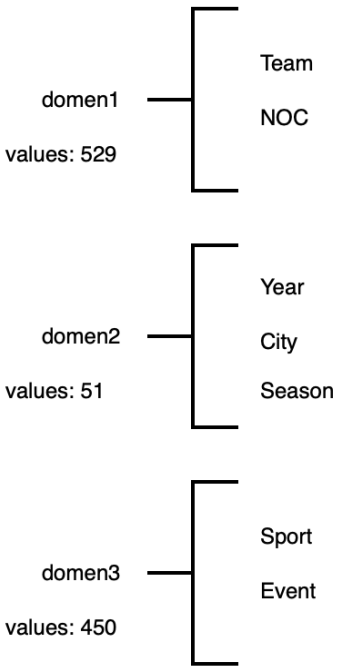

Рисунок 4.5 — Структура таблицы olympic с максимальной длинной домена 3

В отличии от таблицы netflix_titles, домены таблицы olympic не увеличили количество своих значений при увеличении максимальной длины домена. При попытке сжать базу olympic с максимальной длинной домена 4, структура базы никак не поменялось, что говорит о явных корреляциях внутри уже имеющихся доменов.

4.4 Время работы метода

В ходе экспериментов с таблицей olympic были произведены замеры работы времени работы метода в зависимости от длины таблицы и максимальной длины домена, результаты представлены в таблице 4.6

Максимальная длина домена	10000 строк	25000 строк	50000 строк
2	22.77	145.85	381.20
3	331.06	2076.6	5792.79
4	1397.83	8285.38	-

Таблица 4.6 — Зависимость времени сжатия в секундах от длины таблицы и максимальной длины домена в таблице olympic

Как видно из таблицы, увеличение длины домена ведет к внушительному увеличению работы метода, но не всегда это приводит к увеличению процента сжатия, что выявляет нецелесообразность избыточной длины домена

4.5 Время обработки sql-запросов

Так же в ходе экспериментов были произведенны замеры среднего времени выполнения запросов к сжатой базе. Результаты этих замеров показаны в таблице 4.7

В ходе эксперимента было выявленно, что преобазование запросов занимает сравнительно небольшое время, кроме случаев, когда необходим обход по всем словарям. Так, к примеру самым долгим запросом для преобразования оказался UPDATE, в виду того, что для это нужно не просто пройтись по всему словарю, но и выполнить измнения каждой строки, которая подходит под условие для изменения.

Тип запроса	Исходная таблица	Сжатая таблица
Выборка сжатого столбца	0.1248	0.1687
Выборка несжатого столбца	0.1220	0.1647
Выборка с условием	0.0086	0.3923
Вставка строки	0.0053	0.0162
Удаление строки	0.0057	0.6652
Изменение строки	0.019	1.019

Таблица 4.7 — Время обработки запроса в секундах

Заключение

В результате данной работы были решены следующие задачи:

а) был проведен анализ предметной области;
б) был проведен анализ существующих методов сжатия баз данных;
в) была разработана модификация методоа фрактального поиска, позволяющая работать со сжатыми данными;
г) метод был реализован в программном продукте.

В результате отладки и эксплуатации было установленно:

— метод успешно сжимает базы данных с наличием корреляций между атрибутами;
— при максимальной длине домена выше 3 на больших базах метод работает крайне долго;
— при объеме сжимаемых данных, равном половине объема всей таблицы, процент сжатия варьируется от 71.83 до 73.68.

Так же были выявленны следующие недостатки:

— ограниченность в функционале sql-запросов;
— малый объем сообщений об ошибке;
— время работы на больших данных;

Данный алоритм можно продолжить модифицровать:

а) сделать его полнотью параллельным;
б) оптимизировать время работы путем получения структуры оптимальной коллекции доменов из части таблицы;
в) усовершенствовать преобразование sql-запросов с целью расширения функционала.

Список использованных источников

1. Т.Ю. Лымарь, Т.С. Мантрова, Н.Ю. Староверова. Алгоритм фрактального поиска в реляционных базах данных / Т.Ю. Лымарь, Т.С. Мантрова, Н.Ю. Староверова // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика"— Том 3, № 4 (2014).

2. Дейт, К.Дж. Введение в системы баз данных, 8-е издание. / К.Дж. Дейт — М.: Издательский дом "Вильямс 2006. — 1328 с.

3. Apache Parquet [Электронный ресурс]. Режим доступа:https://parquet.apache.org/documentation/latest/.

4. Материал из статьи "Как использовать Parquet и не поскользнуться"[Электронный ресурс]. Режим доступа: https://habr.com/ru/company/wrike/blog/279797/, свободный.

5. Деменюк, С. Л. Просто Фрактал/ С. Л. Деменюк — Санкт-Петербург: Издательство "Страта 2014. — 180 с.

6. Мандельброт Б. Фрактальная геометрия природы / Мандельброт Б. — Москва: Институт компьютерных исследований, 2002. — 656 стр.

7. Смирнов М.А. Обзор применения методов безущербного сжатия данных в СУДБ [Электронный ресурс]— Режим доступа:https://compression.ru/download/articles/db/smirnov_2003_database_compression_review.pdf, свободный.

8. Никульская, Н. А. Задачи, модели и методы Data Mining // XLIII Краевая научная студенческая конференция по математике и компьютерным наукам, сборник материалов [Электронный ресурс]. — Красноярск: Сибирский федеральный ун-т, 2010. — Режим доступа: :http://elib.sfu-kras.ru/bitstream/handle/2311/20642/Nikul%27skaya_%282%29.pdf, свободный.

9. Kaggle [Электронный ресурс]. Режим доступа:https://www.kaggle.com.

10. Vertica VM User Guide [Электронный ресурс]. - Режим доступа: https://www.vertica.com/docs/VMs/Vertica_CE_VM_User_Guide.pdf, свободный.

11. Колоночная база данных [Электронный ресурс]. - Режим доступа: https://wiki.loginom.ru/articles/columnar-database.html, свободный.

12. Реляционная модель данных [Электронный ресурс]. - Режим доступа: https://ru.bmstu.wiki/\T2A\CYRR\T2A\cyre\T2A\cyrl\T2A\cyrya\T2A\cyrc\T2A\cyri\T2A\cyro\T2A\cyrn\T2A\cyrn\T2A\cyra\T2A\cyrya_\T2A\cyrm\T2A\cyro\T2A\cyrd\T2A\cyre\T2A\cyrl\T2A\cyrsftsn_\T2A\cyrd\T2A\cyra\T2A\cyrn\T2A\cyrn\T2A\cyrery\T2A\

`cyrh`, свободный.