

Введение

Объемы данных, собираемых и анализируемых людьми, постоянно растут. Кроме того, постоянно повышаются требования к скорости чтения/записи этих данных. Сами данные обычно хранятся в структурированном виде, обладая рядом характеристик, по которым необходимо осуществлять их выбор. К решению этой проблемы можно подходить несколькими способами: во-первых, масштабировать систему хранения данных — добавление вычислительных мощностей должно снижать время обработки и поиска. Однако, в реальности такое масштабирование не является выгодным — оно стоит денег: сначала покупка оборудования, затем его поддержка.

Глава 1. Введение

1.1 Постановка задачи

Задачи хранения и обработки большого числа данных встречаются повсеместно. При этом хочется делать это эффективно и быстро.

Представим себе торговую площадку, каждый товар обладает рядом характеристик непрерывных или дискретных, как например на рисунках 1.1.1, 1.1.2, 1.1.3.




	<p>Ноутбук Lenovo V130 15</p> <p>5.0</p> <p>Покупателям нравится долгое время работы, небольшой вес</p> <p>Процессор: Core i3 Объем жесткого диска: 500 ГБ Диагональ экрана: 15.6 " Видеокарта: Intel HD Graphics 520 Вес: 1.8 кг</p> <p>28 человек купили этот товар</p>	<p>Есть акции</p> <p>от 16 270 Р</p> <p>251 предложение от 16 270 Р</p>	<p>Показать всё</p> <p>Тип</p> <p><input type="checkbox"/> ноутбук <input type="checkbox"/> ноутбук-планшет <input type="checkbox"/> трансформер</p> <p>Размер экрана</p> <p><input type="checkbox"/> 11"-11.9" <input type="checkbox"/> 12"-12.9" <input type="checkbox"/> 13"-13.9" <input type="checkbox"/> 14"-14.9" <input type="checkbox"/> 15"-15.9" <input type="checkbox"/> 17" и более <input type="checkbox"/> до 11"</p> <p>Разрешение экрана</p> <p><input type="checkbox"/> 1920x1080 <input type="checkbox"/> 1366x768 <input type="checkbox"/> 1600x900 <input type="checkbox"/> 3840x2160 <input type="checkbox"/> 2560x1600</p> <p>Показать всё</p> <p>Процессор</p> <p><input type="checkbox"/> Core i5 <input type="checkbox"/> Core i7 <input type="checkbox"/> Core i3 <input type="checkbox"/> Pentium <input type="checkbox"/> Celeron</p> <p>Показать всё</p>
<p>Выбор покупателей</p> 	<p>Ноутбук Xiaomi Mi Notebook Air 13.3" 2018</p> <p>5.0 7 отзывов</p> <p>Покупателям нравится экран, мощный процессор</p> <p>Объем жесткого диска: 256 ГБ Диагональ экрана: 13.3 " Видеокарта: NVIDIA GeForce MX150 Вес: 1.3 кг Оптический привод: DVD нет</p> <p>410 человек купили этот товар</p>	<p>от 54 400 Р</p> <p>30 предложений от 54 400 Р</p>	
<p>Выбор покупателей</p> 	<p>Ноутбук Xiaomi Mi Gaming Laptop</p> <p>5.0 3 отзыва</p> <p>Покупателям нравится мощный процессор, экран</p> <p>Диагональ экрана: 15.6 " Видеокарта: NVIDIA GeForce GTX 1060 Вес: 2.7 кг Оптический привод: DVD нет 4G LTE: нет</p> <p>398 человек купили этот товар</p>	<p>от 69 000 Р</p> <p>18 предложений от 69 000 Р</p>	

Рисунок 1.1.1 — Продажа электроники

Правильно организованная работа с данными увеличивает скорость выполнения запросов и позволяет экономить на оборудовании. Для этого стоит выбрать правильную структуру данных для хранения. Далее будут рассмотрены типы индексов, используемые в современных СУБД и проведено их сравнение.

Легковые автомобили

Все Новые С пробегом Сохранить поиск

Марка Модель Поколение +

Кузов Коробка Двигатель Привод Объем от, л до

Год от до Пробег от, км до Цена от, ₽ до

Все параметры Показать 52 532 предложения

Audi	2675	Ford	2956	LADA (BA3)	3780	Opel	1989
BMW	4061	Hyundai	2276	Mercedes-Benz	4293	Toyota	2056
Chevrolet	1910	Kia	2339	Nissan	2693	Volkswagen	3432

до 50 000 ₽ до 100 000 ₽ до 150 000 ₽ до 200 000 ₽ до 250 000 ₽ до 300 000 ₽ до 350 000 ₽ до 400 000 ₽ < >

Рисунок 1.1.2 — Продажа автомобилей

1.2 Математическая модель

Сформулируем математическую модель, а также характеристики, на которых будет сделан акцент в данной работе.

В базовом случае интересующий нас объект — это хранилище данных 1.2.1, обрабатывающее пользовательские запросы. Запросы могут быть нескольких типов: создание, чтение, обновление и удаление записей. В зависимости от типа запроса запрашивающая сторона (ей может быть пользователь или какая-то другая система) должна получить некоторый результат: данные, удовлетворяющие условию запроса, если это чтение, и «подтверждение» факта, что данный запрос успешно выполнен. Важна также корректность результатов наших запросов — если наш запрос содержит некоторые условия, то при его работе не должны возвращаться, изменяться и удаляться записи, не удовлетворяющие данному условию. Также мы не должны иметь побочных эффектов в виде появления записей, не создаваемых напрямую с помощью запросов или не предусмотренных внутренней задокументированной логикой работы данной базы данных.

Конечно, обычно СУБД предлагают более широкий список возможностей: поддержку групп и ролей, разграничение доступа, создание и использование

Квартира

Комнаты

Цена

+ Еще фильтры

Найти

Город

Метро

Район

Шоссе

Все

Новостройка

Вторичка

Сохранить мой поиск

Опубликовано

За месяц

Цена

За всю площадь

Общая площадь (м²)

От

До

Площадь кухни (м²)

От

До

Жилая площадь (м²)

От

До

Материал здания

☐ Кирпич
☐ Блоки
☐ Монолит
☐ Панель
☐ Дерево

Год постройки

С

По

Дома под снос

Неважно

Этаж

От

До

Не последний

Этажей в доме

От

До

До метро (по карте)

-

Продавец

Любой

Агент/агентство

Собственник

Показать только

С фотографиями

Добавить ключевое слово

Например: окна во двор, консьерж, название ЖК или адрес

Исключить ключевое слово

Например: окна на улицу, требует ремонта или балкона нет

Рисунок 1.1.3 — Продажа недвижимости



Рисунок 1.2.1 — Схематическое представление системы хранения данных

пользовательских функций, но это не будет рассмотрено, поскольку не имеет непосредственного отношения к рассматриваемой области.

Не имеет особой разницы форма запроса и ответа на него — предполагаем, что существует некоторый протокол, который и используется при общении с данной СУБД при этом на коммуникацию между системами тратится сравнимое и меньшее время, чем на выполнение запроса.

Получаемые результаты могут достаточно сильно зависеть от платформы, на которой будут запускаться тесты. Более подробное описание среды тестирования, платформы и самих тестов будет приведено в следующих главах.

1.3 Оценка полученных результатов

Оценка полученных результатов — результат проведения нескольких тестов. Для начала, считаем, что одна запись — вектор, который содержит числа и строки. При этом осуществлять поиск планируется не по всем значениям, а только по их части. Тестирование должно проводиться для нескольких размеров записей: 10/100 байт, 1/10/100 килобайт, 1 мегабайт. Количество полей, по которым осуществляется поиск стоит взять следующими: 1 (для сравнения со стандартными решениями), 2, 3 (гео-данные), 5, 20, 50. На первом этапе стоит проводить вставку 1/10/100 миллионов записей и замерять количество вставок за единицу времени, а также зависимость этого времени от количества уже вставленных данных. Следующий этап — чтение. Для вышеописанных конфигурация измеряется количество чтений за одну секунду. Финальный этап наиболее приближен к реальной жизни: должны выполняться запросы обоих типов в следующем соотношении — 80/20.

Глава 2. Анализ предметной области

2.1 Типы индексов

B-Tree

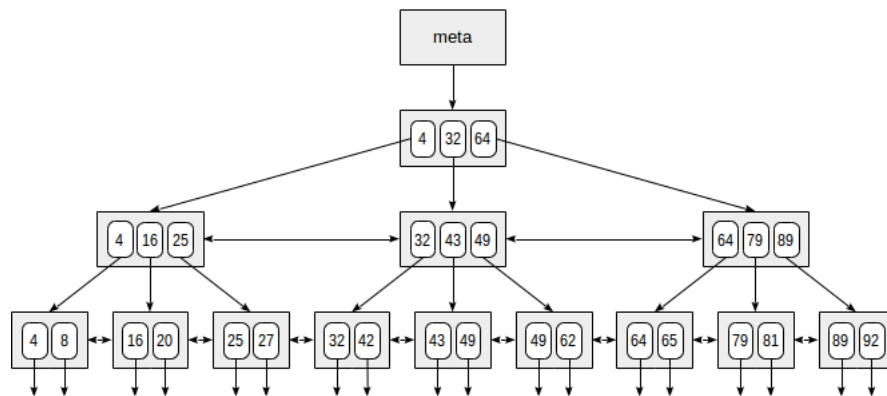


Рисунок 2.1.1 — Схематичный пример индекса по одному полю с целочисленными ключами

Для хранения данных в отсортированном виде используется B-Tree. Чтобы примерно представить себе работу можно вспомнить обычное бинарное дерево (в плане логарифмического времени поиска и структуры). Однако в данном случае всё устроено сложнее: дерево сбалансировано и сильно ветвистое — каждый узел обычно имеет более двух потомков [2.1.1](#).

Поддерживаются операторы сравнения и равенство. Однако для поиска по многомерным структурам данных данный тип индекса не будет подходить — фильтром можно будет использовать лишь для одного ключа, для фильтрации по другим характеристикам придется прибегнуть к перебору, что сильно снижает скорость запроса.

Тем не менее для большинства задач B-дерево всё-таки является хорошим вариантом. B-Tree можно назвать самым популярным индексом, используемым в большинстве современных СУБД как реляционных, так и нереляционных. Существует много модификаций B-Tree: B+Tree (используется в CouchDB, MongoDB), SB-Tree (OrientDB).

Hash

Hash-индекс работает не с индексируемыми ключами, а с их хэшами. Идея хеширования состоит в том, чтобы значению любого типа данных сопоставить некоторую битовую последовательность фиксированной длины. Такое сопоставление называют хеш-функцией. Полученное значение можно использовать как индекс обычного массива, куда и складывать ссылки на строки таблицы.

Это следующий по популярности индекс. Главным недостатком является то, что в запросах поддерживается только операция равенства. Кроме того, возникают коллизии (когда одному хэшу соответствует несколько значений) и трудности с хранением null-значений.

В отличие от других вариантов хранится не само значение, а его хэш, что делает этот индекс компактным и быстрым.

LSM-Tree

Вместо B-дерева данные можно хранить в структуре, называемой LSM-деревом — *Log-structured merge-tree*. Ключевым отличием является то, что в узлах дерева хранятся не сами данные, а операции с ними.

key	lsn	Op code	Value
1	176	REPLACE	2018-05-07 15:00:01
1	53	INSERT	2017-12-31 23:59:01
2	174	REPLACE	2018-05-06 00:00:00
3	175	REPLACE	2018-05-07 09:04:19
3	9	REPLACE	2017-01-01 19:25:43
3	7	INSERT	2017-01-01 19:22:16
4	173	DELETE	
4	168	INSERT	2018-05-05 07:40:01

Рисунок 2.1.2 — Устройство одного уровня в LSM-дереве

При этом если В-дерево целиком хранится на диске, то LSM-Tree допускает возможность частичного хранения в оперативной памяти — L0-уровень (zero level). Все операции вставки делаются в L0, как только место в оперативной памяти заканчивается, данные начинают сбрасываться на диск.

Считается, что LSM-деревья работают быстрее для частых вставок и редких чтений, в отличие от В-деревьев.

Inverted index

Индекс, использующаяся для полнотекстового поиска. Содержит список всех уникальных слов и ссылки на документы, в которых эти слова встретились.

Полнотекстовые запросы выполняют лингвистический поиск в текстовых данных путем обработки слов и фраз в соответствии с правилами конкретного языка: разбиение на слова, отсекаание окончаний, выбор однокоренных слов и т.д. Отдельными задачами при полнотекстовом поиске являются ранжирование результатов запроса и исключение ненужных слов.

Реализации полнотекстового поиска варьируются в различных СУБД. Инвертированный индекс используется в Microsoft SQL Server, MySQL, OrientDB и поисковом движке Elasticsearch.

Обобщением данного типа индекса является *GIN* (Generalized Inverted Index), реализованный в PostgreSQL. Кроме полнотекстового поиска, является подходящим для индексирования массивов и JSON. Обобщенным он называется, потому что операция над индексируемым объектом задается отдельно в отличие от, например, В-Tree, где все операции сравнения уже заданы. В качестве операции могут использоваться такие как «содержит», «пересекается», «содержится».

Количество текстовой информации, окружающей нас огромно: новости, книги, письма и т.д. Для индексирования содержимого этот тип индекса является подходящим. Однако работа с текстом не входит в поставленную задачу.

Пространственные индексы

Большинство современных СУБД имеют типы, предназначенные для работы с пространственными типами данных: точки, прямые, окружности и другие геометрические объекты. Для данных объектов используются свои стратегии индексирования.

Известными решениями является использование пространственной сетки (spatial grid), дерева квадрантов (quadtree) и R-Tree.

Данные индексы используются графовыми базами данных (Neo4j, AllegroGrath), однако существуют специальные дополнения и расширения для известных СУБД, но предназначенные для обработки исключительно пространственной информации — PostGIS, Oracle Spatial, GeoAPI в Redis.

R-Tree

Эта структура данных разбивает многомерное пространство на множество иерархически вложенных и, возможно, пересекающихся, прямоугольников и гиперкубов (для многомерных структур) **2.1.3**.

Подходит для поиска объектов в 2-3-мерном пространстве. Идея лежащая в основе индекса — группировка объектов в зависимости от расстояния друг до друга. Это ускоряет поиск, однако происходит потеря точности, и возвращенный результат может не быть абсолютно точным.

Данный тип индекса поддерживается некоторыми движками СУБД MariaDB (SPATIAL INDEX), PostgreSQL (RTREE), Oracle.

Существует несколько модификаций R-Tree: R+-Tree, R*-Tree. Обобщением R-Tree является X-Tree, который позволяет индексировать данные произвольных размерностей.

Другое обобщение *GiST* (*The Generalized Search Tree*) — обобщенное дерево поиска. Реализовано в PostgreSQL и подобно GIN поддерживает индексирование произвольной информации (геоданные, тексты, изображения и т.д.) с использованием операций «принадлежит», «содержит», «совпадает», «соответствует».

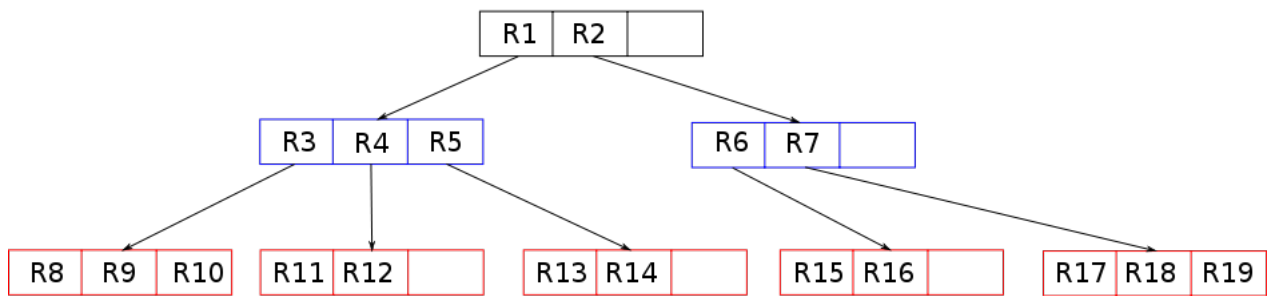
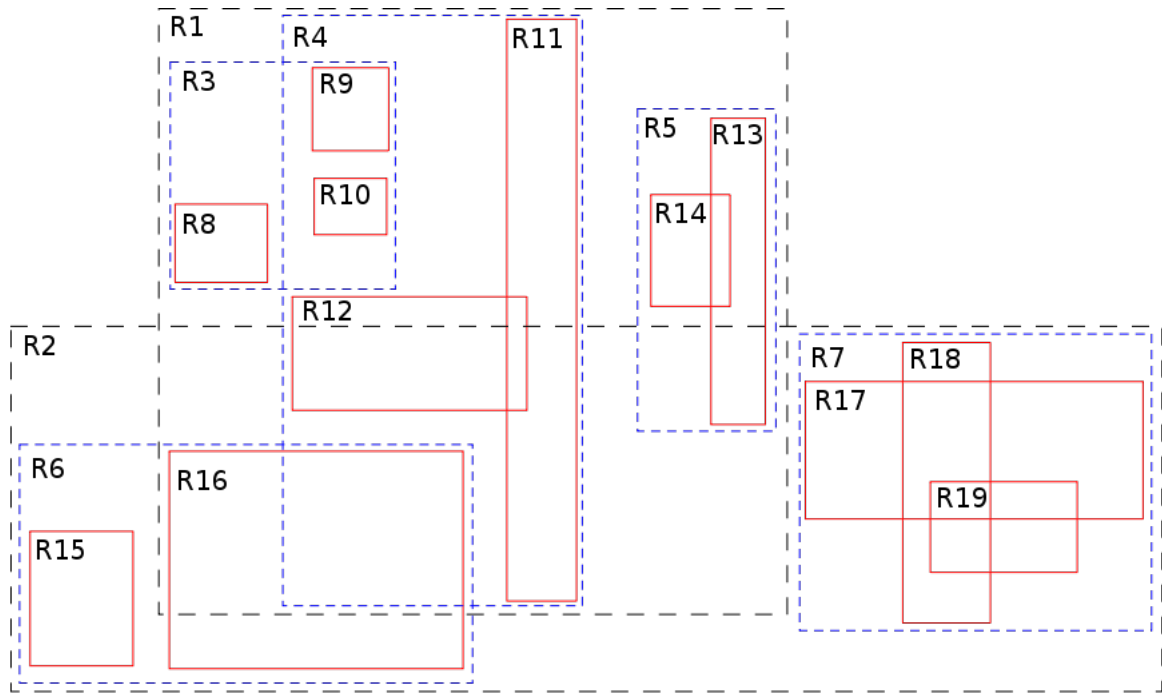


Рисунок 2.1.3 — Пример организации хранения данных в R-Tree

Z-order curve (Кривая Мортон)

Индекс используемый для хранения многомерных данных в одномерной структуре. К каждому значению применяется преобразование Мортон, заключающееся в чередовании двоичных цифр координатных значений, полученный результат называется Z-последовательностью (Z-order curve). Для обработки одномерных данных используется обычное B-дерево.

Позволяет эффективно производить поиск по интервалам значений, однако часть возвращаемого результата может и не находиться в указанном интервале (рис. 2.1.5), поэтому при запросе приходится применять дополнительные механизмы для фильтрации данных. Это накладывает некоторые

	x_1	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
y : 0	000	000000 000001	000100 000101	010000 010001	010100 010101	100000 100001	100100 100101	110000 110001	110100 110101
1	001	000010 000011	000110 000111	010010 010011	010110 010111	100010 100011	100110 100111	110010 110011	110110 110111
2	010	001000 001001	001100 001101	011000 011001	011100 011101	101000 101001	101100 101101	111000 111001	111100 111101
3	011	001010 001011	001110 001111	011010 011011	011110 011111	101010 101011	101110 101111	111010 111011	111110 111111
4	100	100000 100001	100100 100101	101000 101001	101100 101101	110000 110001	110100 110101	111000 111001	111100 111101
5	101	100010 100011	100110 100111	101010 101011	101110 101111	110010 110011	110110 110111	111010 111011	111110 111111
6	110	101000 101001	101100 101101	110000 110001	110100 110101	111000 111001	111100 111101	111110 111111	111111 111111
7	111	101010 101011	101110 101111	110010 110011	110110 110111	111010 111011	111110 111111	111111 111111	111111 111111

Рисунок 2.1.4 — Построение Z-последовательности

ограничения на целесообразность применения данного индекса. Запросы должны быть.

- **Часто задаваемыми.** Распространена практика, когда часть параметров запроса не задается, а остается открытой, однако в данном случае это может серьезно влиять на производительность.
- **«Избирательным».** Границы, устанавливаемые при запросе должны исключать большие объемы данных. Для неравномерно распределенных логических значений пространство поиска может быть сильно увеличено.

	x_1	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
y : 0	000	000000 000001	000100 000101	010000 010001	010100 010101	100000 100001	100100 100101	110000 110001	110100 110101
1	001	000010 000011	000110 000111	010010 010011	010110 010111	100010 100011	100110 100111	110010 110011	110110 110111
2	010	001000 001001	001100 001101	011000 011001	011100 011101	101000 101001	101100 101101	111000 111001	111100 111101
3	011	001010 001011	001110 001111	011010 011011	011110 011111	101010 101011	101110 101111	111010 111011	111110 111111
4	100	100000 100001	100100 100101	101000 101001	101100 101101	110000 110001	110100 110101	111000 111001	111100 111101
5	101	100010 100011	100110 100111	101010 101011	101110 101111	110010 110011	110110 110111	111010 111011	111110 111111
6	110	101000 101001	101100 101101	110000 110001	110100 110101	111000 111001	111100 111101	111110 111111	111111 111111
7	111	101010 101011	101110 101111	110010 110011	110110 110111	111010 111011	111110 111111	111111 111111	111111 111111

Рисунок 2.1.5 — Поиск значений в интервале

При реализации Z-адрес рассматривается исключительно как битовая последовательность. Это значит, что единственное ограничение на тип ключа —

возможность упорядочивания при переходе к двоичному представлению. В итоге доступные типы ограничиваются не только целыми числами, но и числами с плавающей точкой, строками, временными метками...

Данный тип индексирования используется в TransBase[ramsak2000integrating], Accumulo, HBase [nishimura2011md], DynamoDB[DynamoZorderP1; DynamoZorderP2].

Стоит отметить, что данное преобразование не является единственным для отображения многомерных данных в одномерные. Могут использоваться кривые Гильберта или Пеано. Однако Z-последовательность гораздо проще для вычисления.

Индексы с использованием машинного обучения

Можно выделить несколько подходов, которые могут быть использованы для поиска информации и выделения закономерностей в больших массивах данных — Latent Semantic Indexing (LSI) и Hidden Markov Model (HMM). Данные варианты хоть и являются интересными и полезными в некоторых сферах, но примеров их использования в каких-либо СУБД нет.

2.2 Используемые индексы в различных СУБД

СУБД	Индексы
PostgreSQL	B-Tree, R-Tree, Hash, GiST, SP-GiST, GIN, RUM, BRIN, Bloom
MySQL/MariaDB	B-Tree, Hash, R-Tree, Inverted Index
Oracle	B-Tree, B-Tree-cluster, Hash-cluster, Reverse key, Bitmap
MongoDB	B-Tree, Geohash, Text index, Hash
OrientDB	SB-Tree, Hash, Lucene Fulltext, Lucene Spatial
MemSQL	SkipList, Hash, Columnstore

2.3 Выбор платформы для разработки

В качестве платформы, для которой будет реализован индекс на основе кривой Мортонa была выбрана СУБД Tarantool — СУБД с открытым исходным кодом и активно разрабатываемая в настоящее время.

Предлагаемое решение достаточно просто может быть встроено в существующую кодовую базу — около 400 тысяч строк на языке C и 200 тысяч строк на языке Lua.

Основные особенности СУБД Tarantool:

- Удовлетворение принципу ACID
- Наличие двух движков для хранения данных на диске (vinyl) и в оперативной памяти (memtx)
- Хранение данных в формате MessagePack
- Обработка транзакций в одном потоке
- Не только база данных, но и сервер приложений

Реализовываемый индекс будет работать с движком Memtx — все данные будут храниться в оперативной памяти. Часть уже существующих решений может быть переиспользована, например, B+*-Tree (BPS, в терминологии Tarantool).

В СУБД Tarantool достаточно необычная система типов. При работе как с сервером приложений пользователь использует язык программирования Lua 5.1 (LuaJIT 2.1 beta 2) и его типы: "string", "number" (double-precision floating-point), "boolean", "table" (ассоциативный массив), "nil" (отсутствие значения), а так же пользовательские типы данных, представленные типами "userdata" и "cdata".

Уровнем ниже находится MessagePack, обладающий большим количеством типов, более близким к представленным в других языках программирования — unsigned int, signed int, float, double, ... С этим уровнем также взаимодействуют коннекторы, позволяющие внешним системам работать с данными.

Ещё ниже следует система типов Tarantool, обладающая типами которые могут сочетать в себе сразу несколько MessagePack типов, например, "number" — double для нецелых чисел, signed integer для отрицательных целых чисел и unsigned integer для положительных чисел.

При реализации необходимо будет решить несколько инженерных задач:

- Разработка пользовательского интерфейса для работы с индексом
- Встраивание решения в существующую кодовую базу
- Адаптация решения под существующую систему типов

Элементарной единицей является кортеж (tuple), аналог строки в реляционных базах данных. Однако в отличие от реляционных БД кортеж может иметь произвольную длину и содержать произвольные типы. Строгая типизация требуется только для индексируемых полей.

Глава 3. Реализация

Реализацию индекса можно декомпозировать на несколько частей:

- Битовый массив (bit array)
- Логика z-order curve
- Встраивание в движок БД
- Lua-frontend

3.1 Bit array

Как описано в предыдущих пунктах, индексируемое значение получается в результате перемешивания битов указанных индексируемых полей. Данную структуру будем называть битовый массив. При реализации прототипа было решено найти и использовать готовую реализацию на языке C. Реализация была найдена — <https://github.com/noporpoise/BitArray>, она удовлетворяла необходимым функциональным потребностям, однако имела достаточно небольшое покрытие тестами, содержала баги, которые пришлось исправить — <https://github.com/noporpoise/BitArray/pull/15> и была достаточно неоптимальной с учетом специфики решаемой задачи.

С учетом того, что для хранения и представления большинства типов используется 64 бита, размер массива всегда будет кратен 64 элементам. А именно $N * 64$, где N — количество частей в индексе. Приведенная выше реализация содержала достаточно большое число проверок и занимала больше памяти из-за того, что являлось массивом общего назначения с возможностью динамического изменения размера. Для достижения максимальной эффективности пришлось реализовать свой битовый массив постоянного размера и более оптимально работающий с памятью (для выделения памяти использовалось 2 системных вызова, один — выделение структуры со служебными полями, другой сам массив). В полученной реализации создание массива — одна аллокация. Служебное поле "количество элементов в массиве" также потеряло смысл и было удалено поскольку вычисляется как $N * 64$.

Следующий шаг в оптимизации — «векторизация». Большинство современных процессоров поддерживает так называемые векторные инструкции, служащие для обработки массивов данных (SIMD — Single Instruction Multiple Data). По словам разработчиков использование таких инструкций позволяет получать прирост производительности до 30%. О том, что какой-то цикл может быть векторизован можно с помощью директивы `#pragma simd`, однако это не дает гарантий, что цикл будет векторизован, компилятор может и проигнорировать данную директиву, к тому же большинство современных компиляторов могут автоматически находить векторизуемые циклы. Посмотреть за тем, векторизуется цикл или нет, можно с помощью специальных опций компилятора.

Циклы для операций AND и OR были векторизованы для компиляторов GCC v7.4.0 и Apple Clang v11.0.0.