Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет

имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Разработка СУБД с использованием эффективных способов журналирования и хранения информации

|  |  |
| --- | --- |
| Студент ИУ9-62Б | Зворыгин А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_  (ф.и.о.) (подпись, дата) |
| Руководитель курсового проекта | Вишняков И.Э. \_\_\_\_\_\_\_\_\_\_\_\_\_ |

(ф.и.о.) (подпись, дата)

Москва, 2020г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc74418647)

[1 Исследование 5](#_Toc74418648)

[1.1 Основные термины 5](#_Toc74418649)

[1.2 Области применения 7](#_Toc74418650)

[1.3 Существующие реализации 7](#_Toc74418651)

[1.3.1 СУБД Oracle 8](#_Toc74418652)

[1.3.2 СУБД Microsoft SQL Server 10](#_Toc74418653)

[1.3.3 СУБД PostgreSQL 13](#_Toc74418654)

[1.3.4 СУБД Cassandra 15](#_Toc74418655)

[1.4 Постановка задачи 19](#_Toc74418656)

[2 Проектирование 21](#_Toc74418657)

[2.1 Общая архитектура 21](#_Toc74418658)

[2.2 Сетевой интерфейс 23](#_Toc74418659)

[2.3 Процессор запросов 24](#_Toc74418660)

[2.4 Интерпретатор 24](#_Toc74418661)

[2.5 Ядро СУБД 24](#_Toc74418662)

[3 Реализация 27](#_Toc74418663)

[3.1 Межсервисная коммуникация 27](#_Toc74418664)

[3.2 Интерпретатор 27](#_Toc74418665)

[3.3 Хранилище данных 29](#_Toc74418666)

[3.3.1 Оперативная память 29](#_Toc74418667)

[3.3.2 Энергонезависимая память 31](#_Toc74418668)

[3.3.3 Логирование 32](#_Toc74418669)

[3.3.4 Запись данных 33](#_Toc74418670)

[4 Тестирование 35](#_Toc74418671)

[4.1 Тестирование таблиц сортированных строк 35](#_Toc74418672)

[4.2 Тестирование хранилища данных 35](#_Toc74418673)

[4.3 Тестирование интерпретатора 35](#_Toc74418674)

[4.4 Интеграционное тестирование 36](#_Toc74418675)

[ЗАКЛЮЧЕНИЕ 37](#_Toc74418676)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc74418677)

ВВЕДЕНИЕ

Сегодня работа любого веб-сервиса в той или иной степени зависит от эффективности средств хранения информации. Поиск необходимых пользователю данных занимает наибольшую часть времени ответа веб-сервиса. Наиболее популярным средством хранения и организации информации являются СУБД.

Целью этой работы является изучение способов проектирования и реализации высоконагруженных СУБД. Для этого необходимо определить эффективный способ хранения и журнализации данных, реализовать его и сравнить с существующими реализациями СУБД.

# 1 Исследование

В данном разделе будет рассмотрены основные термины, использующиеся в проектировании современных СУБД, области их применения, анализ используемых технологий и определение требований к разрабатываемой СУБД.

## 1.1 Основные термины

*База данных* – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами моделирования данных [1, 2].

*Система управления базами данных* (СУБД) – совокупность программных методов, обеспечивающих управление базами данных. В современной СУБД используются следующие основные компоненты [3, 4, 5].

* ядро СУБД
* компилятор языка БД
* подсистему поддержки времени исполнения
* сервисные программы

*Ядро СУБД* – основной компонент СУБД, отвечающий за управление данными во внешней и внутренней памяти, управление транзакциями и журнализацию [2, 5].

*Внешней памятью* называется энергонезависимая память, используемая для хранения данных. Наиболее используемыми типами памяти являются жесткий диск и флэш память [5].

*Внутренней памятью* зачастую считается оперативная память (энергозависимая память) [2, 5].

*Транзакция* – это последовательность операций над БД, рассматриваемых СУБД как единое целое [5, 6]. Либо транзакция успешно выполняется, и СУБД фиксирует изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД [1, 7].

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти.

Основной функцией *компилятора языка БД* [3, 4] является компиляция операторов языка БД в некоторую выполняемую программу. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением *подсистемы поддержки времени выполнения*, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные *утилиты БД* [2, 8] обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с обращением к компонентам ядра.

*LSM-дерево* (англ. Log-structured merge-tree — журнально-структурированное дерево со слиянием) [9, 10] — используемая во многих [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94) [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), предоставляющая быстрый доступ по [индексу](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) в условиях частых запросов на [вставку](https://ru.wikipedia.org/w/index.php?title=%D0%92%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)&action=edit&redlink=1) (например, при хранении [журналов транзакций](https://ru.wikipedia.org/wiki/%D0%96%D1%83%D1%80%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%82%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D0%B9)). LSM-деревья, как и другие деревья, хранят пары «ключ — значение». LSM-дерево поддерживает две или более различные структуры, каждая из которых оптимизирована под устройство, в котором она будет храниться. Синхронизация между этими структурами происходит блоками.

*B⁺-дерево* [11, 12] — [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) на основе [B-дерева](https://ru.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), сбалансированное n-арное дерево поиска с переменным, но зачастую большим количеством потомков в узле. B⁺-дерево состоит из корня, внутренних узлов и листьев, корень может быть либо листом, либо узлом с двумя и более потомками.

## 1.2 Области применения

СУБД могут быть разделены по характеру обращения к данным.

*Преимущественно запись* (англ. Write-mostly) [6, 7] – изменение данных происходит регулярно, система активно используется пользователями, основным требованием к такой СУБД является оперативное выполнение запросов на изменение данных (добавление, удаление, обновление) – зачастую требуется выполнение запросов в реальном времени (англ. real-time). Например, банковские системы, социальные сети или транспортные системы.

*Преимущественно чтение* (англ. Read-mostly) [6, 7] – изменение данных происходит редко, система используется в основном для долговременного хранения данных и приоритет отдается запросам на получение данных(чтение). Например, веб-сервис Википедия, системы хранения документов – данные паспортов или видео-хостинги. Основными требованиями к таким базам данных является надежность и отказоустойчивость.

*Смешанное* (англ. Mixed) [6, 7] – сочетание обоих подходов к использованию баз данных, например мессенджеры – требуется как быстрый доступ к данным, так и быстрый отклик на отправку сообщений.

Зачастую высоконагруженные СУБД используется в случае преимущественной записи и смешном обращении к данным.

## 1.3 Существующие реализации

В настоящее время существует несколько популярных реализаций высоконагруженных СУБД. Особенное внимание уделяется технологиям хранения данных в этих СУБД.

### 1.3.1 СУБД Oracle

Oracle Database — это объектно-реляционная СУБД (система управления базами данных), созданная компанией Oracle [13].

Сервер Oracle включает базу данных Oracle и экземпляр. Экземпляр состоит из буферов памяти, образующих системную глобальную область (англ. System Global Area - SGA), и фоновых процессов, которые контролируют и выполняют большую часть невидимой работы при выполнении экземпляра.

Экземпляр простаивает (англ. idle) до момента его старта. При запуске читается файл параметров инициализации и на его основе конфигурируется экземпляр. Пользователи могут соединяться с базой данных после того, как экземпляр запустится и база данных будет открыта.

Основные структуры памяти, связанные с экземпляром Oracle.

1. Системная глобальная область (англ. System Global Area- SGA), разделяемая всеми серверными и фоновыми процессами
2. Программная глобальная область (англ. Program Global Area - PGA), частная для каждого серверного и фонового процесса; для каждого процесса выделяется одна PGA.
3. Системная глобальная область (англ. SGA) — это область разделяемой памяти, в которой содержатся данные и управляющая информация экземпляра.

SGA содержит следующие структуры данных:

1. **Кэш буферов БД (**англ. **Database buffer cache)** - для блоков данных, выбираемых из БД.
2. **Журнальный буфер (**англ. **Redo log buffer)** - для кэширования информации повторного выполнения (используемой при восстановлении экземпляра) до момента их записи в журнальные файлы.
3. **Разделяемый пул (**англ. **Shared pool)** - для кэширования различных структур, которые могут совместно использоваться пользователями.
4. **Большой пул (**англ. **Large pool)** - необязательная область, в которой отводится память для буферов, требуемых большими операциями ввода/вывода.
5. **Java-пул**, используемый для Java-кода сеансов и данных внутри виртуальной Java-машины (англ. Java Virtual Machine - JVM).

При запуске экземпляра с помощью Enterprise Manager или SQL\*Plus выводится информация о памяти, выделенной для SGA. В рамках динамической инфраструктуры SGA можно без остановки экземпляра менять размеры кэша буферов БД, разделяемого пула, большого пула, Java-пула и пула потоков.

Преконфигурированная база данных уже настроена и использует подходящие параметры распределение оперативной памяти. Однако по мере роста базы данных может возникнуть необходимость внести изменения в эти параметры.

Oracle позволяет выдавать сигнальные сообщения (alerts) для своевременного определения проблем, связанных с размером структур памяти, и содержит советчики (advisors), которые помогают установить подходящие значения для параметров.

Программная глобальная область (англ. PGA) — это область памяти, выделяемая для каждого серверного процесса, содержащая данные и управляющую информацию этого процесса. Серверный процесс — это процесс, который обрабатывает запросы клиента. Каждый серверный процесс имеет свою приватную область PGA, которая создается при старте серверного процесса. Доступ к этой области имеет только этот серверные процесс, чтения и запись в эту область выполнятся через код Oracle, вызываемый из этого серверного процесса.

Совокупный размер памяти, выделяемый под области PGA и их содержимое, зависит от того, сконфигурирован ли в экземпляре режим разделяемого сервера. Обычно PGA содержит:

* приватную область SQL (англ. Private SQL area), в которой находятся информация привязки и структуры памяти, создаваемые при выполнении команд. Каждый сеанс, в котором выполняется команда SQL, имеет приватную область SQL.
* память сеанса (англ. Session memory), выделяемая для обработки переменных сеанса и другой связанной с сеансом информацией.

### 1.3.2 СУБД Microsoft SQL Server

Microsoft SQL Server [3, 14] — [система управления реляционными базами данных (РСУБД)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94), разработанная корпорацией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft). Основной используемый язык запросов — [Transact-SQL](https://ru.wikipedia.org/wiki/Transact-SQL). Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

Экземпляр Microsoft SQL Server включает в себя системные базы данных (master.model, msdb, tempdb), содержащие служебную информацию, и пользовательские базы данных. Каждая база данных размещается в отдельных файлах – минимум двух: один для самой базы данных – файл данных (mdf-файл), и один для журнала транзакций (ldf-файл). Первый файл данных (mdf-файл) является основным и кроме самих данных содержит системную информацию, второй и все последующие файлы данных являются вторичными (ndf-файлами) и содержат непосредственно сами данные. Расположение этих файлов можно указать при создании базы данных. Основной единицей хранения данных является страница. SQL Server выполняет чтение и запись данных постранично. Вся база данных логически подразделена на страницы, нумеруемые начиная с 0. Размер страницы составляет 8 Кбайт (128 страниц на один мегабайт). Для более эффективного управления страницами они объединяются в экстенты – по 8 страниц в экстенте. Экстенты могут быть двух типов (см. рисунок 1)

* mixed – страницы, входящие в такой экстент, могут принадлежать разным объектам;
* uniform- экстент содержит станицы, принадлежащие одному объекту.

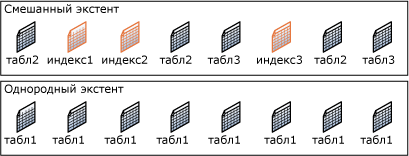


Рисунок 1 – Схема экстентов

Системная информация о странице хранится в заголовке, под который отводится первые 96 байт. Эта информация содержит номер страницы, тип страницы, количество свободного пространства, и ID объекта, владеющего страницей. В конце каждой страницы располагается таблица смещения строк (см. рисунок 2).

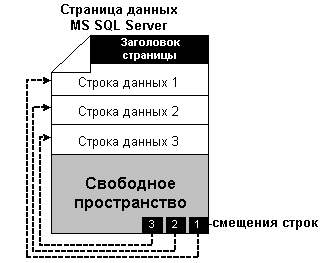


Рисунок 2 – Схема страницы данных

SQL Server использует в файлах данных следующие типы страниц:

* Data - станица содержит строки всех данных за исключением данных типа text, ntext, image, nvarchar(max), varchar(max), varbinary(max) и xml-данных;
* Index - станица содержит информацию о индексах;
* Global Allocation Map - станица данного типа содержит информацию об используемости экстентов (на одной странице хранятся данные об используемости 64000 экстентов);
* Shared Global Allocation Map - станица данного типа содержит информацию об используемости экстентов типа Mixed;
* Page Free Space - страница содержит информацию о количестве свободного пространства на странице;
* Index Allocation Map - страница содержит данные, какие экстенты имеют страницы, принадлежащие одному объекту владельцу;
* Bulk Changed Map - станица содержит информацию об экстентах, измененных посредством набора операций, выполненных после последней операции копирования базы данных (BACKUP LOG);
* Differential Changed Map - станица содержит информацию об экстентах, измененных с момента последней операции копирования базы данных (BACKUP DATABASE).
* Text/Image страница для хранения:
  + o следующих типов LOB-объектов: text, ntext, image, nvarchar(max), varchar(max), varbinary(max) и xml-данных;
  + o столбцов переменной длины, чей размер строки превышает 8 KB: varchar, nvarchar, varbinary и sql\_variant

Далее создается схема, которая отображает разбиения разбиваемых таблиц или индексов на группы файлов (см. рисунок 3).

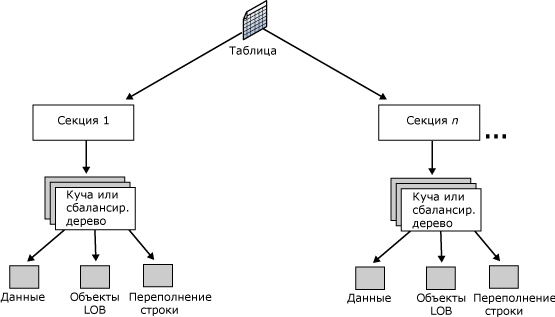


Рисунок 3 – Схема размещения данных

### 1.3.3 СУБД PostgreSQL

СУБД PostgreSQL использует для своих баз данных реляционную модель, поддерживая стандартный язык запросов SQL [3, 4].

У экземпляра СУБД имеется общая для всех серверных процессов память. Большую ее часть занимает буферный кэш (shared buffers), необходимый для ускорения работы с данными на диске. Обращение к дискам происходит через операционную систему (которая тоже кэширует данные в оперативной памяти). PostgreSQL полностью полагается на операционную систему и сам не управляет устройствами. В частности, он считает, что вызов fsync() гарантирует попадание данных из памяти на диск. Кроме буферного кэша в общей памяти находится информация о блокировках и многое другое; через нее же серверные процессы общаются друг с другом. У каждого серверного процесса есть своя локальная память. В ней находится кэш каталога (часто используемая информация о базе данных), планы запросов, рабочее пространство для выполнения запросов и другое (см. рисунок 4).

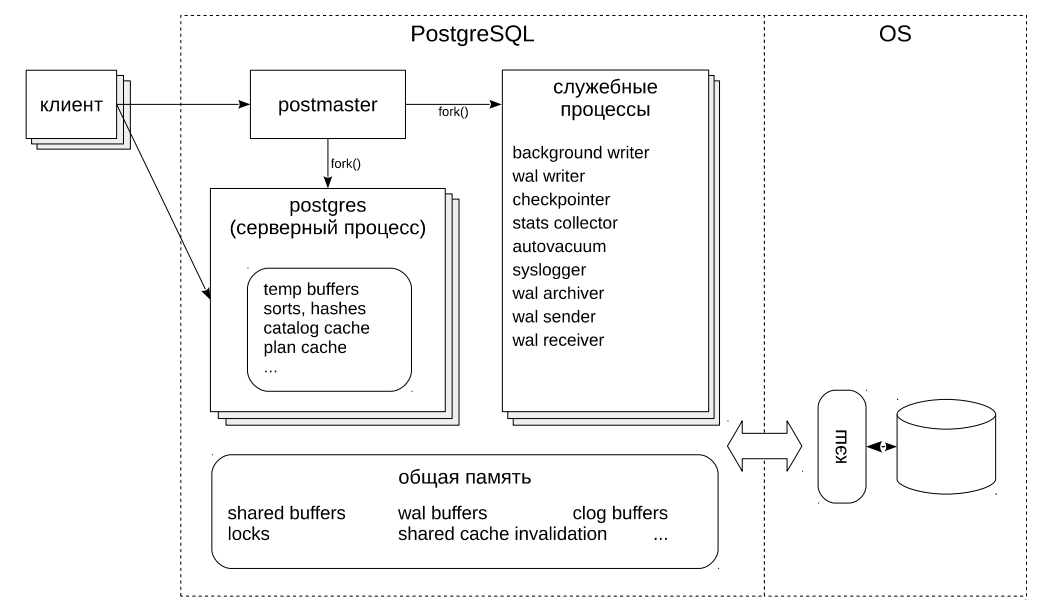


Рисунок 4 – Схема PostgreSQL

Экземпляр СУБД работает с несколькими базами данных. Эти базы данных называются кластером. Хранение данных на диске организовано с помощью табличных пространств. Табличное пространство указывает расположение данных (каталог на файловой системе). Оно может использоваться несколькими базами данных. Например, можно организовать одно табличное пространство на быстрых дисках для активно использующихся данных и другое — на медленных дисках для архивных данных. Объекты (таблицы и индексы) хранятся в файлах; каждый объект занимает один или несколько (2-3) файлов внутри каталога табличного пространства. Кроме того, файлы разбиваются на части по 1 ГБ. Необходимо учитывать влияние потенциально большого количества файлов на используемую файловую систему (см. рисунок 5).

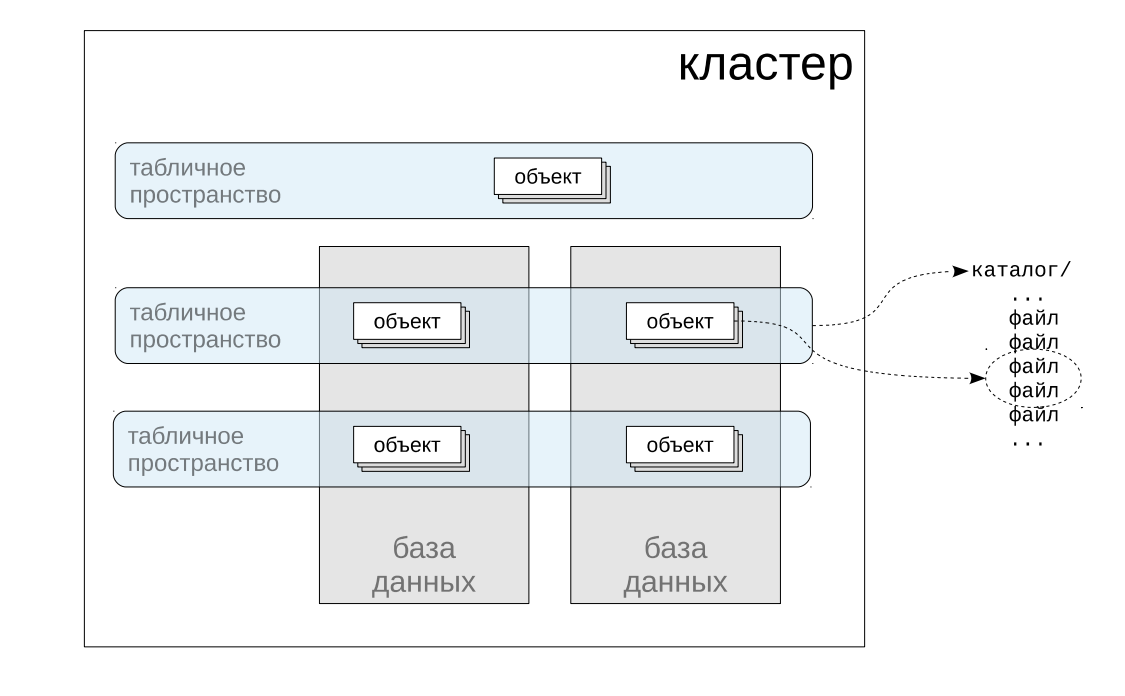


Рисунок 5 – Схема кластера

Буферный кэш используется для сглаживания скорости работы памяти и дисков. Он состоит из массива буферов, которые содержат страницы (блоки) данных и дополнительную информацию об этих страницах. Размер страницы обычно составляет 8 КБ, хотя может устанавливаться при сборке. Буферный кэш, как и другие структуры в памяти, защищен блокировками от одновременного доступа. Блокировки организованы достаточно эффективно, чтобы не создавать большой конкуренции. Любая страница, с которой работает СУБД, попадает в кэш. Часто используемые страницы остаются в кэше надолго; редко используемые — вытесняются и заменяются другими страницами.

### 1.3.4 СУБД Cassandra

Cassandra [15] – NoSQL СУБД, но для проектирования SQL СУБД необходимо рассмотреть и NoSQL СУБД, определить различия и причины той или иной архитектуры.

Таблица в памяти существует для каждого колоночного семейства и позволяет запомнить значение моментально. Технически это хеш-таблица (англ. [hashmap](http://en.wikipedia.org/wiki/Hashmap)) с возможностью одновременного доступа(англ. [concurrent access](http://en.wikipedia.org/wiki/Concurrent_access)) на основе структуры данных, называемой “списками с пропусками” (англ. [skip list](http://en.wikipedia.org/wiki/Skip_list)) [5]. Журнал закрепления один на всё пространство ключей и сохраняется на диске. Журнал представляет собой последовательность операций модификации. Так же он разбивается на части при достижении определённого размера.

Такая организация позволяет сделать скорость записи ограниченной скоростью последовательной записи на жесткий диск и при этом гарантировать долговечность данных (англ. [data durability](http://en.wikipedia.org/wiki/ACID#Durability)) [6]. Журнал закрепления в случае аварийного останова узла читается при старте сервиса кассандры и восстанавливает все таблицы в памяти. Получается, что скорость упирается во время последовательной записи на диск, а у современных жёстких дисков это порядка 100МБ/с. По этой причине журнал закрепления советуют вынести на отдельный дисковый носитель [5, 13].

Рано или поздно память может заполниться. Поэтому таблицу в памяти также необходимо сохранить на диск. Для определения момента сохранения существует ограничение объёма занимаемыми таблицами в памяти (англ. [memtable\_total\_spacein\_mb](http://www.datastax.com/docs/1.1/configuration/node_configuration#memtable-total-space-in-mb)), по умолчанию это ⅓ максимального размера кучи Java (англ. [Java heapspace](http://en.wikipedia.org/wiki/Java_Virtual_Machine_heap#Heap)) [15]. При заполнении таблицами в памяти объёма больше чем это ограничение, кассандра создает новую таблицу и записывает старую таблицу в памяти на диск в виде сохраненной таблицы (англ. [SSTable](https://git-wip-us.apache.org/repos/asf?p=cassandra.git;a=blob_plain;f=src/java/org/apache/cassandra/io/sstable/SSTable.java;h=9a29066b8af7334be8fa9325fa480cee632abca0;hb=refs/heads/cassandra-1.1)). Сохранённая таблица после создания больше никогда не модифицируется (англ. [is immutable](http://en.wikipedia.org/wiki/Immutable)). Когда происходит сохранение на диск, то части журнала закрепления помечаются как свободные, таким образом освобождая занятое журналом место на диске. Нужно учесть, что журнал имеет переплетённую структуру из данных разных колоночных семейств в пространстве ключей, и какие-то части могут быть не освобождены, так как некоторым областям будут соответствовать другие данные, все ещё находящиеся в таблицах в памяти (см. рисунок 6).

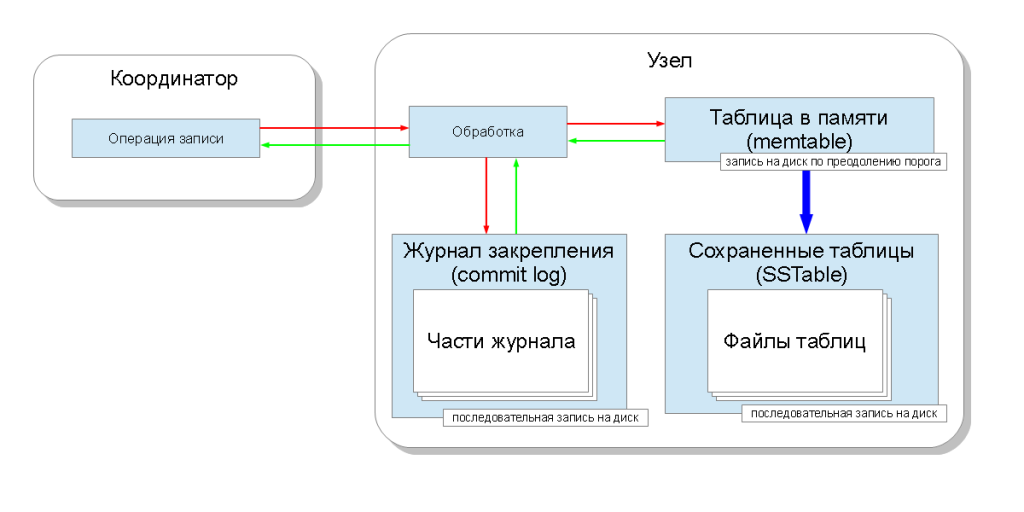


Рисунок 6 – Схема узла

В итоге, каждому колоночному семейству соответствует одна таблица в памяти и некоторое число сохранённых таблиц. Теперь, когда узел обрабатывает запрос чтения, ему необходимо запросить все эти структуры и выбрать самое последнее по метке времени значение. Для ускорения этого процесса существует три механизма: блум-фильтрация (англ. [bloom filter](http://en.wikipedia.org/wiki/Bloom_filter)), кэш ключей (англ. [key cache](http://www.datastax.com/docs/1.1/operations/tuning#tuning-data-caches)) и кэш записей(англ. [record cache](http://www.datastax.com/docs/1.1/operations/tuning#tuning-data-caches)) (см. рисунок 7):

* блум-фильтр — это структура данных, которая занимает немного места и позволяет ответить на вопрос: содержится ли элемент, а в нашем случае это ключ, в множестве или нет. При чем, если ответ — “нет”, то это 100%, а если ответ “да”, то это, возможно, ложноположительный ответ. Это позволяет уменьшить количество чтений из сохранённых таблиц;
* кэш ключей сохраняет позицию на диске записи для каждого ключа, таким образом уменьшая количество операций позиционирования (англ. [seek operations](http://en.wikipedia.org/wiki/Seek_time#Seek_time)) во время поиска по сохранённой таблице;
* кэш записей сохраняет запись целиком, позволяя совсем избавиться от операций чтения с диска.

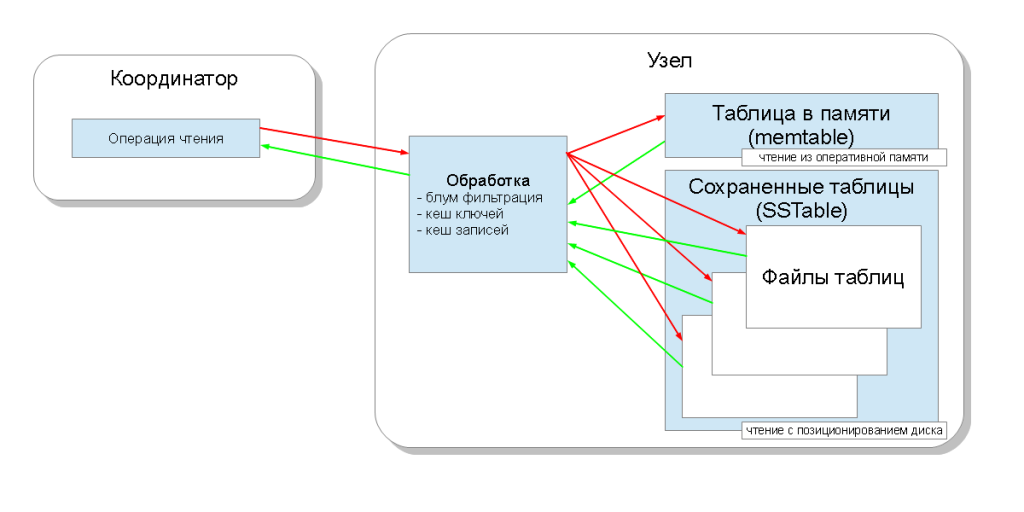


Рисунок 7 – Схема работы узла

В определенный момент времени данные в колоночном семействе перезапишутся — придут колонки, которые будут иметь те же имя и ключ. То есть, возникнет ситуация, когда в более старой сохранённой таблице и более новой будут содержаться старые и новые данные. Для того, чтобы гарантировать целостность, кассандра обязана читать все эти сохранённые таблицы и выбирать данные с последней меткой времени. Получается, что количество операций позиционирования жёсткого диска при чтении пропорционально количеству сохранённых таблиц. Поэтому для того, чтобы освободить перезаписанные данные и уменьшить количество сохранённых таблиц, существует процесс уплотнения (англ. [compaction](http://www.datastax.com/docs/1.1/operations/tuning#tuning-compaction)). Он читает последовательно несколько сохранённых таблиц и записывает новую сохранённую таблицу, в которой объединены данные по меткам времени. Когда таблица полностью записана и введена в использование, кассандра может освободить таблицы-источники (таблицами, которые её образовали). Таким образом, если таблицы содержали перезаписанные данные, то эта избыточность устраняется. Понятно, что во время такой операции объем избыточности увеличивается — новая сохранённая таблица существует на диске вместе с таблицами-источниками, а это значит, что объем места на диске всегда должен быть такой, чтобы можно было произвести уплотнение (см. рисунок 8).

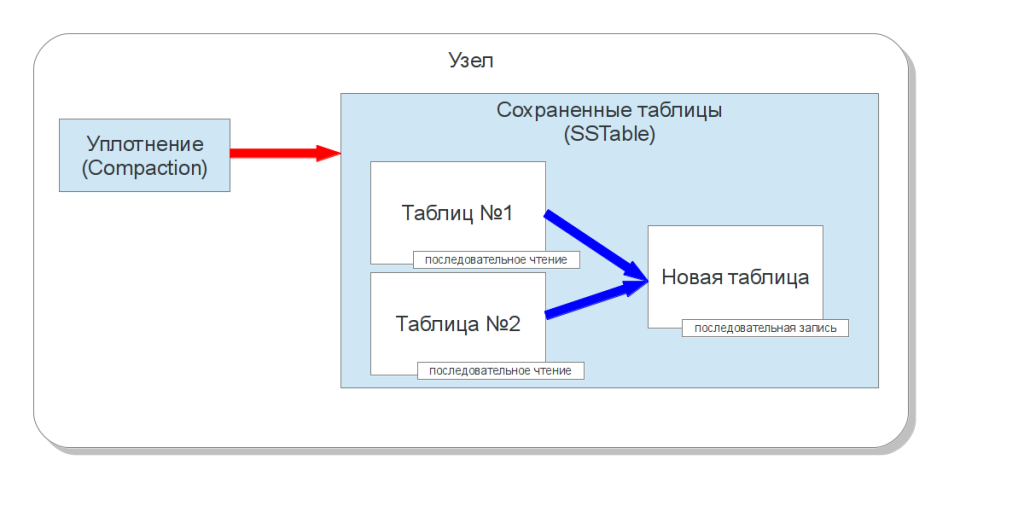


Рисунок 8 – Схема уплотнения

## 1.4 Постановка задачи

Проанализировав существующие реализации и используемые технологии, необходимо спроектировать модель высоконагруженной SQL СУБД. Приоритетным направлением в проектировании была выбрано LSM-Дерево (основная часть Cassandra). Поскольку имеет преимущества перед реализациями SQL СУБД.

* Запись за O(1) – в реализациях SQL СУБД запись происходит как в файлы индексов, так и в файлы данных за O (n ln(n)).
* Чтение за O(ln(n)) – только из одного файла SSTable [10], посредством блум-фильтра, в то время как в SQL СУБД происходит обращение к данным посредством обращения сначала к файлу индексов, а после к файлу с записями.
* Отсутствуют перезаписи в файлы, файлы записываются единожды для многократного чтения – экономия дискового ресурса.

Также при проектировании необходимо определить какими особенностями SQL СУБД необходимо будет пожертвовать, поскольку использование LSM-Дерева серьезно ограничивает реляционную модель данных [5, 9].

# 2 Проектирование

В данном разделе необходимо определить общую архитектуру СУБД и архитектуру важнейших компонент: ядра СУБД, процессора запросов и интерпретатора. Также требуется провести анализ и определить технологии, которые будут использоваться для проектирования отдельных компонент. Код проекта находится в репозитории GitHub [16].

## 2.1 Общая архитектура

Для проектирования такой программной системы необходимо выбрать подход к организации архитектуры, существуют два основных варианта [4, 5]

* монолитная архитектура
* микросервисная архитектура

Необходимо определить для данного конкретного случая – проектирования СУБД плюсы и минусы данных подходов и выбрать наилучший (см. таблица 1).

Таблица 1 – Сравнение видов архитектур

|  |  |  |
| --- | --- | --- |
| Тип архитектуры | Плюсы | Минусы |
| Монолитная | * Ранее связывание позволяет избегать излишних затрат на выполнение кода | * Сложность поддержки кода, требуется контролировать незначительные изменения кода для поддержки работоспособности |
| Микросервисная | * Инкапсуляция задач каждого модуля * Простота поддержки кода благодаря позднему связыванию | * Затраты на многоуровневое выполнение операций * Поддержка многопоточного кода при использовании общих данных |

Для проектирования общей архитектуры был выбран микросервисный подход (см. рисунок 9) [3, 4]

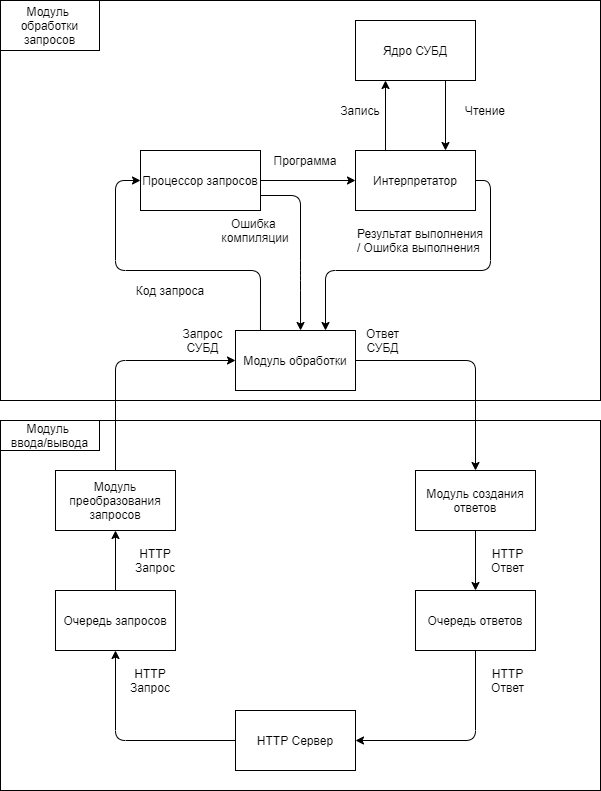


Рисунок 9 – Общая архитектура СУБД

Основными частями архитектуры стали:

* модуль ввода/вывода
* процессор запросов
* интерпретатор
* ядро СУБД

Модуль ввода/вывод работает асинхронно, получает запросы на выполнение запросов в СУБД, передает его в процессор запросов через буфер входных запросов, ожидает выполнения запроса, после получения результата запроса отправляет ответ пользователю.

Процессор запросов принимает запрос к базе данных, передает его в парсер. При лексических и синтаксических ошибках, парсер должен возвращать ошибку компиляции. При удачной компиляции запрос передается в интерпретатор.

Интерпретатор занимается исполнением переданного разобранного запроса передавая команды ядру СУБД. При обнаружении ошибок возвращает ошибку времени исполнения.

Ядро СУБД отвечает за хранение и изменение данных, а также поддержание их целостности, журнализацию и логирование. При экстренном завершении работы отвечает за восстановление данных.

## 2.2 Сетевой интерфейс

В качестве модуля ввода/вывода был разработан HTTP - сервер на базе фреймворка Netty с асинхронной обработкой запросов – запрос (листинг 1) и ответ (листинг 2).

Листинг 1 – Схема запроса

1. POST / HTTP/1.1
2. Host: database.ip
3. Content-Type: text/plain; charset=utf-8
4. Content-Length: length
5. Accept-Language: en-us
6. Accept-Encoding: gzip, deflate
7. Connection: Keep-Alive
9. select \*
10. from scheme.table
11. where id = 1

Листинг 2 – Схема ответа

1. HTTP/1.1 200 OK
2. Date: Mon, 27 Jul 2009 12:28:53 GMT
3. Server: Netty/2.2.14 (Win32)
4. Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
5. Content-Length: 88
6. Content-Type: text/plain
7. Connection: Closed
8. 1 “data” 1 2.0f

## 2.3 Процессор запросов

Процессор запросов при получении запроса для исполнения распознаёт текст запроса с помощью парсера SQL запросов – JSQLParser [18]. Выходными данными парсера является абстрактное синтаксическое дерево.

На данный момент парсер поддерживает все основные диалекты SQL – Oracle, PostgreSQL и Microsoft SQL.

## 2.4 Интерпретатор

Интерпретатор при помощи рекурсивного спуска по абстрактному синтаксическому дереву выполняет запрос. Для каждого типа узлов этого дерева создан обработчик, который обрабатывает определенный этап запроса. Необходимо поддержать минимальный функционал запросов

* create schema <schema\_name>
* create table <schema\_name>.<table\_name> types(…)
* insert into <schema\_name>.<table\_name> values (…)
* select \* from <schema\_name>.<table\_name> where id = n
* delete from <schema\_name>.<table\_name> where id = n

## 2.5 Ядро СУБД

Общая архитектура ядра СУБД представлена на рисунке 10

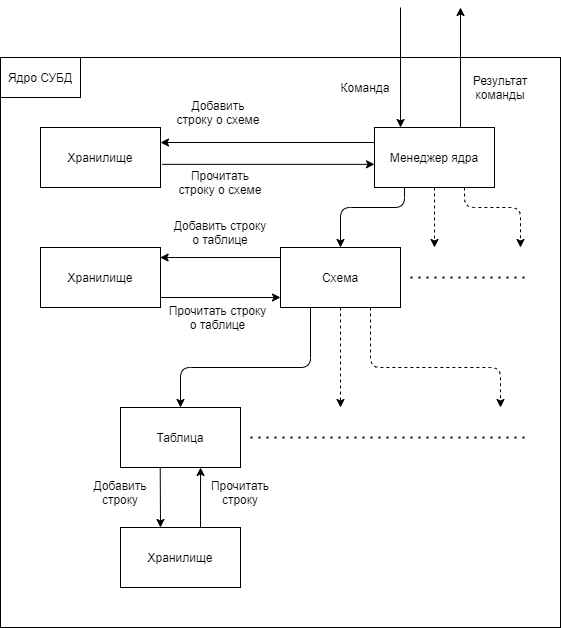


Рисунок 10 – Схема ядра СУБД

В ядре СУБД необходимо поддержать следующие функции для управления данными в базе данных:

* создание схемы
* создание таблицы
* вставка строк
* чтение строк

В необходимо реализовать поддержку стандартных типов данных для хранения в таблице.

* float, double
* int, long
* string

Также ядро СУБД должно поддерживать индексацию по любому первичному ключу [3, 7]. Архитектура ядра СУБД нескольких основных компонент

* Объектное представление схемы
* Объектное представление таблицы
* Объектное представление служебных таблиц

Для каждого объектного представления необходимо поддерживать хранилище для каждого объектного представления. Данное хранилище будет управлять данными во внешней и во внутренней памяти, реализация данного хранилища абстрагирована от других элементов архитектуры.

# 3 Реализация

В данном разделе будут описаны детали реализации отдельных компонент СУБД. Особое внимание уделяется архитектуре и реализации хранилища данных [10].

## 3.1 Межсервисная коммуникация

Для взаимодействия модулей была разработан буфер обмена между сервисами. Благодаря нему данные передаются асинхронно при минимальном количестве конкурентных синхронизаций. Ключевой идеей является создание входного и выходного подбуферов (см. листинг 3) для исключения синхронизаций при малом количестве элементов в буферах, т.е. при наличии всего одного элемента в буфере при конкурентном обращении происходит синхронизация как входных потоков, так и выходных потоков.

Листинг 3 – Буфер данных

1. public class DbStorage<T> extends AbstractDbModule
2. implements IDbStorage<T> {
3. private final Logger logger =
4. LoggerFactory.getLogger(DbStorage.class);
5. private final BlockingQueue<T> inputBuffer;
6. private final BlockingQueue<T> outputBuffer;
7. public DbStorage(BlockingQueue<T> inputBuffer,
8. BlockingQueue<T> outputBuffer) {
9. this.inputBuffer = inputBuffer;
10. this.outputBuffer = outputBuffer;
11. }
12. }

## 3.2 Интерпретатор

Интерпретатор производит обработку синтаксического дерева используя функции хранилища. В качестве примера приводится обработчик select запроса (см. листинг 4).

Листинг 4 – Обработка запроса select

1. public class PlainSelectEngine extends DSQLEngine<PlainSelect> {
2. public PlainSelectEngine(DataStorage dataStorage) {
3. super(dataStorage);
4. }
5. public List<Row> execute(PlainSelect plainSelect)
6. throws RuntimeError {
7. FromItem fromItem = plainSelect.getFromItem();
8. Table table = (Table) fromItem;
9. try {
10. List<Row> rowList =
11. dataStorage
12. .selectRows(SelectSettings.Builder.newBuilder()
13. .setSchemaName(table.getSchemaName())
14. .setTableName(table.getName())
15. .build());
16. return rowList;
17. } catch (DataStorageException dataStorageException) {
18. throw new RuntimeError(dataStorageException.getMessage());
19. }
20. }
21. }

Интерпретатор делегирует запросы ядру СУБД.

## 3.3 Хранилище данных

Хранилище данных основано на создании LSM (Log-structured merge tree) [8, 9, 10] дерева. Общая схема реализации хранилища представлена на рисунке 11.

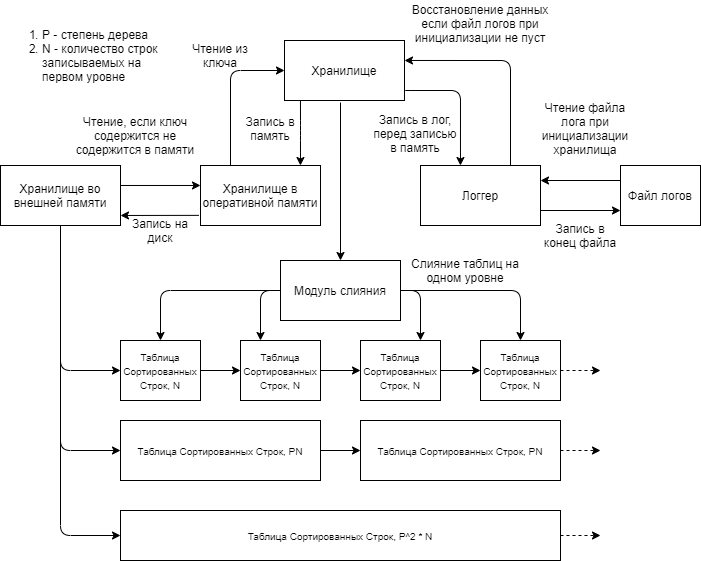


Рисунок 11 – Схема реализации хранилища

### 3.3.1 Оперативная память

В оперативной памяти постоянно находятся индексы и часть выгруженных данных – кэш. В кэше находятся последние используемые данные. Кэш реализован вытесняющий, полностью заменяющий предыдущий кэш. Кэш данных в памяти хранится в виде B+ дерева [11, 12].

При чтении данных, если интересующие данные находятся в кэше, обращение к внешней памяти не происходит. Если данные в кэше не находятся, происходит чтение из внешней памяти, и прочитанные данные помещаются в кэш.

Все чтения с диска происходят только в оперативную память, а чтение самих данных происходит только из памяти. Чтение из внешней памяти происходит блоками – аналогично чтению памяти в процессорах – кэш линиями. Количество блоков, размещенных в памяти настраиваемо.

В памяти постоянно находится индексация блоков LSM дерева в соответствии с уровнями, для каждого блока хранится метаинформация.

Для записи и чтения данных в таблице происходят захваты блокировок – Read-Write Lock.

### 3.3.2 Энергонезависимая память

При большом количестве данных в кэше и при успешном завершении работы СУБД, данные помещаются во внешнюю память. Формат размещения данных в каждом блоке определяется согласно рисунку 12.

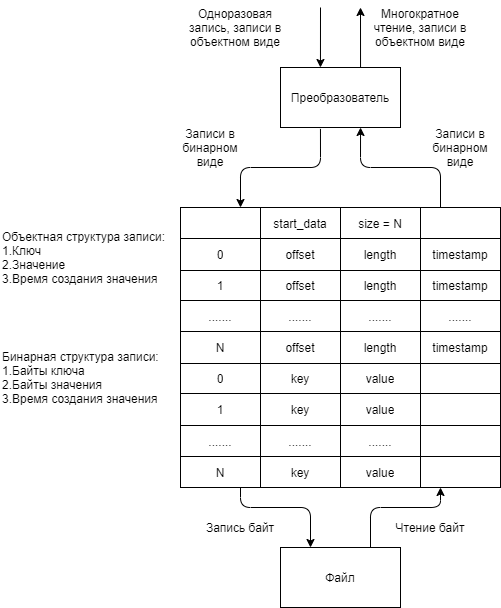


Рисунок 12 - Схема реализации таблицы сортированных строк

Ключи хранят время своего добавления, для определения приоритетного ключа при конкурентном обращении к СУБД.

Таблицы сортированных строк при достижении определенных настраиваемых размеров сливаются в большие таблицы. Во время сливания таблиц удаляются одинаковые ключи (выбирается более новый ключ). Также при достижении нижнего уровня, ключи помеченные удаленными удаляются из таблицы (см. листинг 5).

Листинг 5 - Код слияния таблиц

1. private void merge(int level) {
2. List<SSTable<K, V>> tables = new ArrayList<>(levelMap.get(level));
3. List<Record<K, V>> records = readAllKeys()
4. .entrySet()
5. .stream()
6. .map(entry -> new Record<>(entry.getKey(), entry.getValue()))
7. .collect(Collectors.toList());
8. int i = 0;
9. while (true) {
10. long shift = 64L \* range(level);
11. List<Record<K, V>> recordList =
12. records.stream().skip(i).limit(shift).collect(Collectors.toList());
13. if (recordList.isEmpty()) {
14. break;
15. }
16. if (levelMap.get(level + 1) == null) {
17. levelMap.put(level + 1, new ConcurrentLinkedDeque<>());
18. }
19. SSTable<K, V> ssTable = new SSTable<>(
20. byteConverter,
21. directory.getAbsolutePath(), level + 1,
22. levelMap.get(level + 1)
23. .stream()
24. .map(SSTable::getIndex)
25. .max(Integer::compareTo)
26. .orElse(0) + 1);
27. ssTable.putAll(recordList.toArray(Record[]::new));
28. addTable(level + 1, ssTable);
29. i += shift;
30. }
31. tables.forEach(table -> {
32. levelMap.get(table.getLevel()).remove(table);
33. table.delete();
34. });
35. }

При наличии в таблице большого количества удаленных записей, запускается фиктивное слияние таблиц, для фактического удаления лишних записей. Данными операциями занимаются специальные модули управления внешними таблицами сортированных строк.

### 3.3.3 Логирование

Во избежание потери данных при экстренных завершениях работы СУБД необходимо поддерживать лог операций записи. При успешной записи на диск текущий лог удаляется [9].

При новом запуске СУБД сначала читаются файлы логов и данные помещаются в кэш. Лог всегда заполняется добавлением строки в конец файла – режим append записи файлов, во избежание перезаписи файла (см. листинг 6).

Листинг 6 - Код записи в лог

1. public synchronized void put(@NotNull K key, @NotNull V value) {
2. try {
3. // Append mode
4. FileOutputStream fileOutputStream = new FileOutputStream(file, true);
5. BufferedOutputStream bos = new BufferedOutputStream(fileOutputStream);
6. byte[] keyBytes = byteConverter.keyToBytes(key);
7. byte[] valueToBytes = byteConverter.valueToBytes(value);
8. int allocate = 4 + 4 + keyBytes.length + valueToBytes.length;
9. byte[] allByteArray = new byte[allocate];
10. ByteBuffer buff = ByteBuffer.wrap(allByteArray);
11. buff.putInt(keyBytes.length);
12. buff.put(keyBytes);
13. buff.putInt(valueToBytes.length);
14. buff.put(valueToBytes);
15. count++;
16. bos.write(buff.array());
17. bos.close();
18. } catch (IOException ioException) {
19. throw new WriteToFileException("Cannot append row to logfile: "
20. + Throwables.getStackTraceAsString(ioException));
21. }
22. }

### 3.3.4 Запись данных

Данные записываются в файлы таблиц в определенном формате только один раз. После этого таблицы уже не переписываются. Данные записываются в определенном формате (см. листинг 7).

Листинг 7 - Код записи данных в таблицу

1. public void putAll(@NotNull Record<K, V>[] records) {
2. int size = records.length;
3. int startData = 8;
4. int allocate = 8;
5. int currentOffset = 0;
6. for (Record<K, V> record : records) {
7. byte[] key = byteConverter.keyToBytes(record.getKey());
8. byte[] value = byteConverter.valueToBytes(record.getValue());
9. keyList.add(record.getKey());
10. offsets.add(currentOffset);
11. startData += 4;
12. allocate += 4;
13. lengths.add(key.length);
14. startData += 4;
15. allocate += 4;
16. timestamps.add(record.getTimestamp());
17. startData += 8;
18. allocate += 8;
19. keys.add(key);
20. allocate += key.length;
21. values.add(value);
22. allocate += value.length;
23. currentOffset += key.length + value.length;
24. }
25. meta = new Meta<>(keyList.toArray((int value) -> (K[]) new Key[value]));
26. byte[] allByteArray = new byte[allocate];
27. ByteBuffer buff = ByteBuffer.wrap(allByteArray);
28. buff.putInt(startData);
29. buff.putInt(size);
30. for (int i = 0; i < keys.size(); i++) {
31. buff.putInt(offsets.get(i));
32. buff.putInt(lengths.get(i));
33. buff.putLong(timestamps.get(i));
34. }
35. for (int i = 0; i < keys.size(); i++) {
36. buff.put(keys.get(i));
37. buff.put(values.get(i));
38. }
39. FileOutputStream fileOutputStream = new FileOutputStream(file);
40. BufferedOutputStream bos = new BufferedOutputStream(fileOutputStream);
41. }

# 4 Тестирование

В данном разделе будут описаны методы и результаты отдельных компонент СУБД, также будет проведено комплексное тестирование СУБД.

## 4.1 Тестирование таблиц сортированных строк

Было проведено тестирование таблиц сортированных строк (см. листинг 8).

Листинг 8 – Тестирование таблиц сортированных строк

1. @Test
2. public void writeTest() throws DataStorageException, IOException {
3. LSMFileTree<DefaultKey, Row> lsmFileTree =
4. new LSMFileTree<>(new TableConverter(table,
5. Arrays.asList(Type.INTEGER),
6. table.getTypes()), path);
7. Map<DefaultKey, Row> map = new TreeMap<>();
8. map.put(new DefaultKey(Type.INTEGER, 1),
9. new Row(table, Arrays.asList("Andrey", 20)));
10. map.put(new DefaultKey(Type.INTEGER, 4),
11. new Row(table, Arrays.asList("John", 16)));
12. lsmFileTree.putAll(map);
13. map = lsmFileTree.readAllKeys();
14. assertEquals(map.get(new DefaultKey(Type.INTEGER, 1))
15. .getValues().get(0), "Andrey");
16. assertEquals(map.get(new DefaultKey(Type.INTEGER, 2))
17. .getValues().get(0), "Alex");
18. }

## 4.2 Тестирование хранилища данных

Было проведено тестирование хранилища данных (см. листинг 9).

Листинг 9 – Тестирование хранилища данных

1. @Test
2. public void simpleWriteReadTest() throws Exception {
3. Table table = DSQLTable.Builder.newBuilder()
4. .setName("table")
5. .setTypes(Arrays.asList(Type.STRING, Type.INTEGER))
6. .build();
7. LSMStorage<Key, Row> storage =
8. new LSMStorage<>(new StorageProperties(new TableConverter(table,
9. Arrays.asList(Type.INTEGER),
10. table.getTypes()), "table", path));
11. Row testValue = new Row(table, Arrays.asList("Andrey", 20));
12. storage.init();
13. storage.put(new DefaultKey(Type.INTEGER, 1), testValue);
14. storage.pushToDrive();
15. assertEquals("Andrey", storage.get(
16. new DefaultKey(Type.INTEGER, 1)).getValues().get(0));
17. }

## 4.3 Тестирование интерпретатора

Было проведено тестирование функций интерпретатора (см. листинг 10).

Листинг 10 – Тестирование интерпретора

1. @Test
2. public void writeTest() throws DataStorageException, IOException {
3. DSQLTable table = DSQLTable.Builder.newBuilder()
4. .setName("Table")
5. .setTypes(Arrays.asList(Type.INTEGER, Type.STRING, Type.INTEGER))
6. .setRowToKey(0)
7. .build();
8. table.insertRows(InsertSettings.Builder.newBuilder()
9. .setRows(Arrays.asList(
10. Arrays.asList(1, "Andrey", 20)
11. ))
12. .build());
13. storage.pushToDrive();
14. List<Row> rows = table.selectRows(SelectSettings.Builder.newBuilder()
15. .build());
16. assertEquals(rows.get(0).getValues().get(1), "Andrey");
17. }

## 4.4 Интеграционное тестирование

Было проведено комплексное тестирование с помощью создания конкурентного обращения к СУБД. Результаты тестирования сравнивались с двумя другими СУБД PostgreSQL и ClickHouse при аналогичных обращениях и схемах СУБД. Каждый клиент посылает фиксированное количество запросов. Ниже представлен сравнительный график времени выполнения всех запросов от количества клиентов (см. рисунок 13).

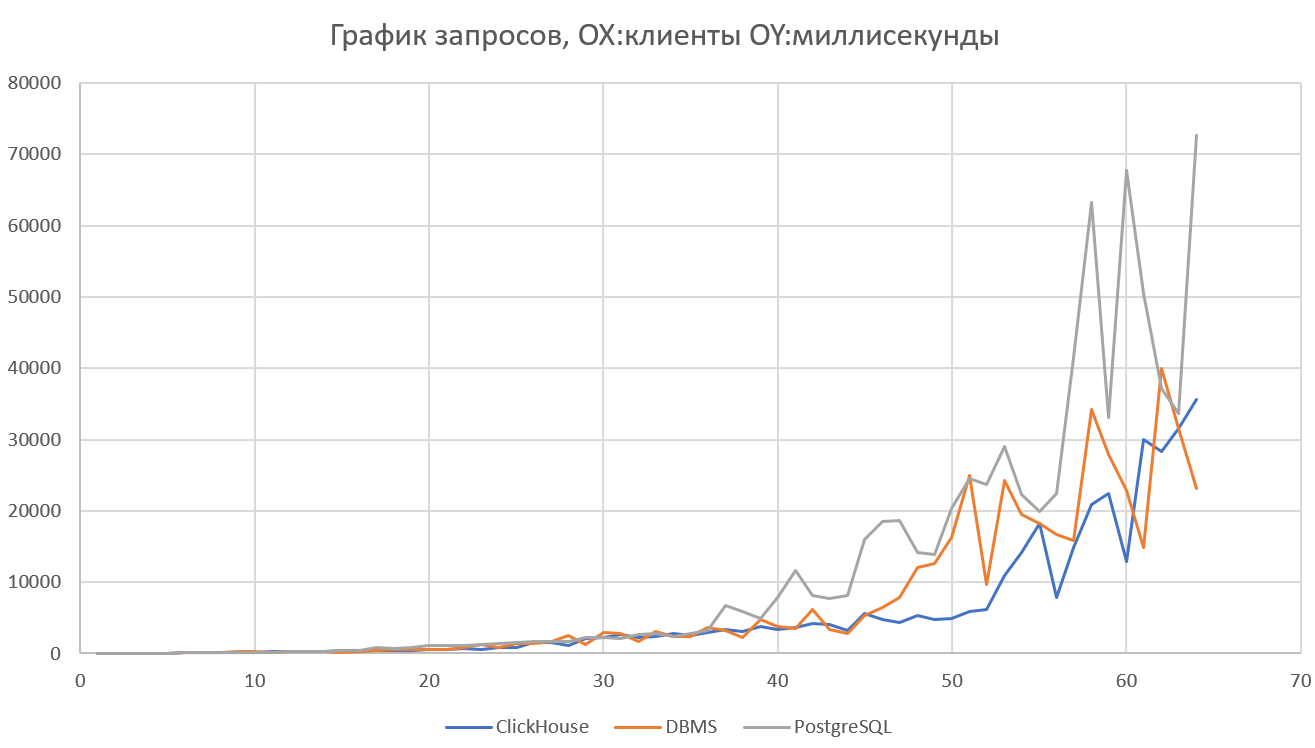


Рисунок 13 – Сравнение результатов

ЗАКЛЮЧЕНИЕ

Была реализована СУБД на основе хранилища информации на базе LSM дерева. Был проведен сравнительный анализ с другими СУБД на основе, которого можно сделать вывод, что хранилище на базе LSM дерева оказалось наиболее эффективным способом хранения информации в высоконагруженных СУБД.

Разумеется, для комплексного анализа данной проблемы в будущем необходимо поддержать полный набор функций SQL СУБД. А именно, поддержка индексов, транзакций и хранимых процедур.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

|  |
| --- |
| 1 Гарсиа-Молина Г. Системы баз данных. Полный курс / Гарсиа-Молина Г., Ульман Д., Уидом Д. – Москва: Издательский дом "Вильямс", 2003. – 1088 с. |
| 2 Дэйт К.Дж. Системы баз данных. Полный курс . - Москва: Издательский дом "Вильямс", 2018. – 1328 с. |
| 3 SQL Fundamentals Databases: сайт. – 2005. – URL: http:/​/​databases.about.com/​od/​sql/​a/​sqlfundamentals.htm (дата обращения: 28.01.2020). |
| 4 Агальцов, В.П. Базы данных. Учебник / В.П. Агальцов. – Москва.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. – 272 c. |
| 5 Кроенке Д., Ауер Д. Database Concepts 3-е издание . – Pearson, 2007. –576 c. |
| 6 Гиампаоло Д. Practical File System Design with the Be File System . – Morgan Kaufmann, 1999. – 256 c. |
| 7 Леман П., Яо Л. ACM Transactions on Database Systems for concurrent operations on B-trees . – Morgan Kaufmann, 1998. – 315 с. |
| 8 Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика . – Москва: Издательский дом "Вильямс", 2003. – 1436 с. |
| 9 Ченг E., Галик Д. The log-structured merge-tree (LSM-tree) // Acta Informatica . – 1996 . – №21 . – 351–385 с. |
| 10 Лео C., Карей M. LSM-based storage techniques: a survey // The VLDB Journal . – 2005 . – №15 . – 161–213 с. |
| 11 Кормен T., Лисерсон C. Introduction to Algorithms . – MIT Press and McGraw-Hill, 2001. – 434–454 c. |
| 12 Кормер А., Дуглас К.. The Ubiquitous B-Tree // Computing Surveys . – 1996 . – 351–385 с. |
| 13 Бауер Р., МакКрейт E. Organization and Maintenance of Large Ordered Indices . – Mathematical and Information Sciences Report. №. 20, 2005 |
| 14 Хальдер Р., Кортези А. Abstract Interpretation of Database Query Languages // Computer Languages, Systems & Structures . – Morgan Kaufmann, 2011. – 123–131 с |
| 15 Карпентер Д., Хьюитт Э. Cassandra. Полное руководство . – 2016. |
| 16 GitHub: сайт. – 2021. – URL: <https://github.com/don-dron/dbms> (дата обращения 03.03.2021) |
| 17 Кнут Д. Сортировка и поиск // Искусство программирования // . – Диалектика-Вильямс, 2019 . – 481–491 с.  18 GitHub: сайт. – 2021. – URL: <https://github.com/JSQLParser/JSqlParser> (дата обращения 03.03.2021) |
|  |