|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное автономное 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 СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

**РЕФЕРАТ**

по дисциплине: «Технологии обработки больших данных»

Студент Журавлев Николай Вадимович

Группа ИУ5-44М

Название Способы межсетевого взаимодействия между различными СУБД

Студент **Журавлев Н.В.**

*подпись, дата фамилия, и.о.*

Преподаватель **Сухобоков А.В.**

*подпись, дата фамилия, и.о.*

Оценка

*Москва, 2025*

**Содержание**

[Введение 3](#_Toc196906550)

[Описание системы 3](#_Toc196906551)

[Ожидаемые преимущества 4](#_Toc196906552)

[Сегодняшнее состояние 4](#_Toc196906553)

[Обзор СУБД, которые включены в систему 4](#_Toc196906554)

[Виды СУБД 4](#_Toc196906555)

[MongoDB 8](#_Toc196906556)

[Neo4j 10](#_Toc196906557)

[Cassandra 11](#_Toc196906558)

[Сравнение с озером данных 11](#_Toc196906559)

[Основная идея построения 12](#_Toc196906560)

[Недостатки Data lake 13](#_Toc196906561)

[Обзор платформ для создания и управления озёр данных 15](#_Toc196906562)

[Определение критических различий 20](#_Toc196906563)

[Описание системы 20](#_Toc196906564)

[Описание взаимодействия модулей 20](#_Toc196906565)

[Синтаксис запросов между разными СУБД 22](#_Toc196906566)

[Классы СУБД 24](#_Toc196906567)

[Парсер запросов 25](#_Toc196906568)

[Возможности применения в других областях 28](#_Toc196906569)

[Заключение 29](#_Toc196906570)

[Список источников 30](#_Toc196906571)

# Введение

В данной работе подробно рассматривается система для взаимодействия пользователя с множеством баз данных, находящимися в различных системах управления базами данных (СУБД). Эта система разработана с целью сделать процесс получения информации более интуитивным и удобным для пользователей, что особенно актуально в условиях растущего объема данных и разнообразия источников информации. Первоначально пользователю необходимо выбрать нужные ему базы данных и соответствующие СУБД, из которых он планирует извлекать данные. Список доступных для выбора СУБД заранее определён разработчиком. Для извлечения данных нужно писать запросы по разработанному синтаксису. Пользователь имеет возможность получать информацию из нескольких баз данных и СУБД одновременно. Кроме того, данная система обладает гибкостью, так как имеет возможность использовать потенциально любую СУБД, при условии, что для этого предварительно добавлен необходимый функционал, соответствующий указанному в статье шаблону. В статье также описывается работа ключевых элементов системы, которые необходимо реализовать для полноценного функционирования будущей системы. Также в статье представлен шаблон, которые необходимо соблюдать для добавления новых СУБД, что делает систему масштабируемой.

# Описание системы

В данной работе разрабатывается системы для взаимодействия пользователя с множеством баз данных, находящимися в различных системах управления базами данных (СУБД).

Первоначально пользователю необходимо выбрать нужные ему базы данных и соответствующие СУБД, из которых он планирует извлекать данные. Список доступных для выбора СУБД заранее определён разработчиком, в данном варианте MongoDB, Neo4j, Cassandra.

Для извлечения данных нужно писать запросы по разработанному синтаксису. Пользователь имеет возможность получать информацию из нескольких баз данных и СУБД одновременно.

Кроме того, данная система должна обладать гибкостью, а именно иметь возможность использовать потенциально любую СУБД, при условии, что для этого предварительно добавлен необходимый функционал, соответствующий разработанному шаблону класса СУБД.

# Ожидаемые преимущества

Эта система разработана для упрощения и повышения удобства работы с информацией, что особенно актуально в условиях стремительного роста объемов данных и их разнообразия. Единая платформа позволяет управлять как структурированными, так и NoSQL-данными, предоставляя пользователям гибкость: в зависимости от задачи можно выбрать наиболее подходящий способ хранения, поскольку разные СУБД эффективны для разных типов данных. Поддержка различных структур хранения в рамках одной системы обеспечивает лучшую доступность и восприятие информации.

# Сегодняшнее состояние

## Обзор СУБД, которые включены в систему

### Виды СУБД

Система предполагает использование любого типа СУБД. Стоит рассмотреть не реляционные СУБД, так как они имеют разнообразные синтаксис, что стоит учитывать при разработке системы.

В настоящее время существует четыре основные модели баз данных: ключ-значение, семейство столбцов, документальные и графовые [1].

Базы данных ключ-значения имеют хранилище ключ-значение. Они позволяют разработчику приложения хранить данные без схемы. Эти данные обычно состоят из строки, которая представляет ключ, и фактических данных, которые считаются значением в отношениях «ключ-значение». Сами данные обычно представляют собой своего рода примитив языка программирования (строка, целое число, массив) или объект, который создаётся привязками языков программирования к хранилищу ключ-значение. Это заменяет необходимость в фиксированной модели данных и делает строгими требования к правильному форматированию данных без данных [2].

Графовая модель базы данных — это модель, в которой структуры данных для схемы и/или экземпляров моделируются как направленный, возможно, помеченный граф или обобщение структуры данных графа, где манипулирование данными выражается с помощью графо-ориентированных операций и конструкторов типов, а соответствующие ограничения целостности могут быть определены в структуре графа [3].

База данных семейства столбцов — это база данных NoSQL, которая хранит данные с использованием столбцового подхода, в отличие от реляционных, которые упорядочивают данные по строкам. Данные, хранящиеся в базе данных семейства столбцов, выбираются вертикально, что делает частичное чтение более эффективным, поскольку загружается только набор атрибутов строки [4].

Вместо хранения данных в фиксированных строках и столбцах базы данных документов используют гибкие документы. Документ – это запись в базе данных документов. Документ обычно хранит информацию об одном объекте и любых связанных с ним метаданных. Документы хранят данные в парах поле-значение. Значения могут быть различных типов и структур, включая строки, числа, даты, массивы или объекты [5].

Так же бывают мультимодельные СУБД. Мультимодельная база данных — это база данных, предназначенная для поддержки нескольких моделей данных в одной системе хранения данных. Это означает, что такая система может хранить, индексировать и запрашивать данные в нескольких моделях. Этот тип базы данных обеспечивает единый интерфейс для обеспечения согласованности, безопасности и доступа к данным, а также устраняет необходимость сложных преобразований и миграций между различными базами данных [6].

Можно выделить 4 метода построения мультимодельных БД: Polyglot persistence, мультимодельные СУБД на основе реляционной модели, мультимодельные СУБД на основе документной модели, СУБД «без основной модели» [7].

1. Polyglot persistence

Одним из самых известных методов построения БД является Polyglot persistence. Этот метод подразумевает выбор столько баз данных, сколько необходимо, чтобы все требования были удовлетворены. Данный метод может быть оптимальным решением, когда необходимо обеспечить обратную совместимость с устаревшим приложением. Новая система баз данных может работать параллельно с устаревшей системой баз данных; хотя устаревшее приложение по-прежнему остается полностью функциональным, новые требования могут быть учтены при использовании новой системы баз данных. Очевидно, следует избегать перекладывания бремени всех этих задач обработки запросов и синхронизации базы данных на уровень приложения, то есть, в конечном итоге, на программистов, которые поддерживают приложения для обработки данных. Вместо этого обычно лучше ввести уровень интеграции. Затем уровень интеграции занимается обработкой запросов: разбивает запросы на несколько подзапросов, перенаправляет запросы в соответствующие базы данных и повторно объединяет результаты, полученные из баз данных, к которым осуществляется доступ. Более того, уровень интеграции должен обеспечивать согласованность между базами данных: он должен синхронизировать данные в различных базах данных путем распространения дополнений, модификаций или удалений между ними. Однако многоязычие сопряжено с серьезными недостатками:

* Отсутствие одинокого способа доступа к данных. Нет уникального интерфейса запроса или языка запросов, и, следовательно, доступ к системам баз данных не унифицирован и требует знания всех необходимых методов доступа к базе данных.
* Согласованность. Согласованность между базами данных является серьезной проблемой, поскольку необходимо обеспечить ссылочную целостность во всех базах данных (например, если запись в одной базе данных ссылается на запись в другой базе данных), а в случае дублирования данных (и, следовательно, в разных представлениях в нескольких базах данных одновременно) дубликаты должны быть обновлены или удалены практически одновременно. Базовые системы баз данных разрабатываются независимо. Более новые версии баз данных могут быть несовместимы с уровнем интеграции, и администратору приходится отслеживать частые обновления.
* Логическая избыточность. Логической избыточности можно избежать только при разработке базы данных, которая строго распределяет непересекающиеся подмножества данных по разным базам данных. Это может противоречить некоторым требованиям пользователей к доступу.
* Безопасность. Уровень интеграции должен обеспечивать контроль доступа, и все подключенные базы данных должны быть настроены так, чтобы разрешать только ограниченный доступ [8].

1. Мультимодельные СУБД на основе реляционной модели

Данный тип включается в себя те СУБД, которые изначально было построены под реляционную модель, но после в не были добавлены Nosql модели.

1. Мультимодельные СУБД на основе документной модели

Данный тип включается в себя те СУБД, которые используют документальную модель как основную, но также имеет поддержку остальных. В качестве основной модели может выбрана любая другая из существующих моделей.

1. СУБД «без основной модели»

Это СУБД, которые не имеют основной модели данных, при этом поддерживает какие-либо другие.

Далее представлены СУБД, поддерживающиеся в системе.

### MongoDB

MongoDB – это не реляционная, а документно-ориентированная система управления базами данных. Документно-ориентированная СУБД заменяет концепцию «строки» более гибкой моделью, «документом». Позволяя использовать вложенные документы и массивы, документно-ориентированный подход дает возможность представлять сложные иерархические отношения с помощью одной записи. Также нет предопределенных схем: ключи и значения документа не имеют фиксированных типов или размеров. Когда нет фиксированной схемы, добавлять или удалять поля по мере необходимости становится проще. MongoDB – СУБД общего назначения, поэтому помимо создания, чтения, обновления и удаления данных она предоставляет большинство тех функций, которые можно ожидать от системы управления базами данных. Специальные типы коллекций и индексов MongoDB поддерживает коллекции данных TTL (time-to-live), срок действия которых должен истечь в определенное время, такие как сеансы и коллекции фиксированного размера, для хранения недавно полученных данных, например, журналов. MongoDB также поддерживает частичные индексы, ограниченные только теми документами, которые соответствуют фильтру критериев, чтобы повысить эффективность и уменьшить необходимый объем дискового пространства [9].

Каждый запрос к БД начинается с “db”, после чего идёт символ точки, а затем название коллекции, к которой необходимо обратиться. Так же возможно использоваться collection('name\_ collection ').

Для простого добавления элемента в коллекцию используется метод insert:

db.collection('name\_ collection ').insert({ “elem\_name” : “elem\_value” });

Для добавления множества элементов необходимо через запятую указать нужные:

db.collection('name\_ collection ').insert({ “arr” : [ { a : 1 , b : 1 } , { a : 2, b : 2 } ] });

Всю информацию из коллекции можно удалить через метод remove (сама коллекция при этом не удаляется):

db.collection('name\_ collection ').remove();

Удалить объект по какому-то параметру необходимо следующим образом:

db.collection('name\_ collection ').remove( { “elem\_name” : “elem\_value” });

Для обновления коллекции используется метод update:

db.collection('name\_ collection ').update( { “elem\_name” : ” elem\_value” });

Но тогда будет произведена замена всего документа, а, чтобы такого не произошло необходимо использовать модификатор “$set”:

db.collection('name\_ collection ').update({ $set: { “elem\_name”: ” new\_elem\_value” }});

Так же с помощью модификатора “$unset”:можно удалить ключ:

db.collection('name\_ collection ').update({ $unset: { “elem\_name”: ” elem\_value” }});

Имеется возможность использования Upsert. При его использовании, если документ по запрашиваемую критерию не найден, то он будет создан, если же найден, то он будет обновлён, как обычно. Чтоб использовать upsert нужно просто в команде update добавить третий параметр равный true:

db.collection('name\_ collection ').update({$set: “elem\_name”: ” new\_elem\_value”}, true);

Для поиска используется Find. Возвращает массив документов в виде коллекции, если документов нет — пустую коллекцию.

db.collection('name\_ collection ').find({ “elem\_name” : ” elem\_value”});

Для добавления условий поиска используется следующие операторы: $lt — меньше, $lte — меньше или равно, $gt — больше, $gte — больше или равно, $ne — не равно.

Так же имеется возможность использовать для поиска регулярные выражения [10].

### Neo4j

Neo4j — это собственная графовая база данных, а это означает, что она реализует настоящую графовую модель вплоть до уровня хранения. Вместо использования «абстракции графика» поверх другой технологии данные хранятся в Neo4j так же, как вы можете доносить свои идеи на доску. Помимо основного графа, Neo4j также обеспечивает транзакции ACID, поддержку кластеров и аварийное переключение во время выполнения. [11]

Для простого добавления элемента в коллекцию используется метод create:

CREATE (node:label { key1: value1, key2: value2, . . . . . . . . . });

Узел можно удалить через метод remove:

MATCH (node {attribute1: 'value1'}) REMOVE node.attribute2 RETURN node;

Для обновления коллекции используется метод update:

MATCH (node {attribute1: 'value1'}) SET node.attribute2='value2' RETURN node.attribute1, node.attribute2;

Имеется возможность использования Upsert. При его использовании, если документ по запрашиваемую критерию не найден, то он будет создан, если же найден, то он будет обновлён, как обычно. Чтоб использовать upsert нужно просто в команде update добавить третий параметр равный true:

db.collection('name\_ collection ').update({$set: “elem\_name”: ” new\_elem\_value”}, true);

Для поиска значений необходимо указать в конце строки return N.

MATCH (n) WHERE (n.id = 0) RETURN n;

### Cassandra

Apache Cassandra — это распределенная база данных NoSQL с открытым исходным кодом, которой доверяют тысячи компаний благодаря своей масштабируемости и высокой доступности без ущерба для производительности. Линейная масштабируемость и проверенная отказоустойчивость стандартного оборудования или облачной инфраструктуры делают его идеальной платформой для хранения критически важных данных [12].

Для простого добавления элемента в коллекцию используется метод insert:

INSERT INTO student (id, citizenship, first\_name, last\_name, age) VALUES (now(), 'Russia', 'Ivan', 'Ivanov', 25);

Запись можно удалить через метод delete:

DELETE FROM student WHERE id=54daf810-9aeb-11ea-b1d1-3148925e06e7;

Для обновления коллекции используется метод update:

UPDATE student SET first\_name = 'Example', last\_name='Example' WHERE

id=54daf810-9aeb-11ea-b1d1-3148925e06e7;

Для поиска используется команда select:

SELECT title, MAX(price) FROM course GROUP BY title;

## Сравнение с озером данных

Основная идея системы может показаться похожей на озера данных. Для определения различий необходимо рассмотреть подробнее.

Озеро данных (Data lake)— это система или хранилище данных, которые хранятся в необработанном формате. Data lake обычно представляет собой единое хранилище данных, включающее необработанные копии данных исходной системы. Data lake может включать структурированные данные из реляционных баз данных (строки и столбцы), полуструктурированные данные (CSV, журналы, XML, JSON), неструктурированные данные (электронные письма, документы, PDF-файлы) и двоичные данные (изображения, аудио, видео) [13].

Озёра данных предназначены для того, чтобы собирать, хранить и обрабатывать большое количество информации, поступающей практически непрерывным потоком. Такую информацию называют Big Data, или большими данными. Data Lake полезны всем компаниям, которые планируют анализировать большие данные любой области. Само по себе озеро данных бесполезно, потому что это просто хранилище. Чтобы с ним работать, нужны инструменты для очистки, структурирования, извлечения и анализа данных, и специалисты для работы с этими инструментами [14].

Часто Data lake используют для хранения важной информации, которая пока не используется в аналитике. Или даже для данных, которые кажутся бесполезными, но, вероятно, пригодятся компании в будущем. Data lake позволяет накапливать данные «про запас», а не под конкретный запрос бизнеса. За счет того, что данные всегда «под рукой», компания может быстро проверить любую гипотезу или использовать данные для своих целей [15].

### Основная идея построения

Основная идея Data lake заключается в следующем: все данные, отправляемые организацией, будут храниться в единой структуре данных, называемой Data Lake. Данные будут храниться в озере в исходном формате. Будет исключена сложная предварительная обработка и преобразование данных при загрузке в Data lake. Как только данные помещены в озеро, они доступны для анализа всем сотрудникам организации.

Озеро представляет собой файловое хранилище на нескольких серверах, в котором лежат данные. Как правило данные распределены между серверами, чтобы хранилище можно было быстро масштабировать — подключить новые серверы для расширения места.

К серверам настраивают подключение разных источников данных, доступных компании. Каналы поставки данных называют пайплайнами, а всю схему подключения — ETL-процессом. Обычно всё настроено так, чтобы данные загружались автоматически.

Хотя Data lake и неструктурированное, порядок в нём всё-таки должен быть, иначе спустя время накопится огромное количество данных, в которых невозможно будет разобраться. Поэтому перед добавлением в озеро данные размечают и запоминают, откуда и в каком формате они поступили. В итоге внутри озера данных хранятся не только сами объекты, но и метаданные, то есть информация об объектах. Это облегчает поиск, извлечение и анализ данных в будущем.

В архитектуре озера данных должны быть предусмотрены инструменты резервного копирования, чтобы информация не терялась.

Общий алгоритм работы выглядит следующим образом:

1. В одном из источников формируются данные.
2. По заранее настроенному маршруту данные с серверов отправляются в Data lake.
3. При поступлении данные размечаются: записывается их источник, время поступления, формат и структура.
4. Данные помещаются в озеро и хранятся там. Как правило, срок хранения не ограничен, хотя иногда данные удаляют по мере устаревания или использования в аналитике.

При необходимости данные извлекают из хранилища по определённым критериям и используются [15, 16].

### Недостатки Data lake

Озера данных оптимизированы для высокой пропускной способности, но ради этого приходится жертвовать качеством данных [17]:

1. В Data lake не требуется структурировать данные, поэтому их сложнее анализировать.
2. Data lake не имеет инструментов для целостного получения всех данных.
3. Без квалифицированного контроля за озерами данных трудно гарантировать конфиденциальность и безопасность хранилища.
4. Если управление озером организовано плохо, в нем быстро накапливаются большие объемы неконтролируемых, и, возможно, бесполезных данных. Для эффективной фильтрации данных и отсечения недостоверных источников требуется высокая квалификация.

Если в Data lake хранится слишком много данных, которые плохо организованы, без надлежащего управления метаданными и надежного управления данными, найти соответствующие данные становится все труднее. Через определенное время данные теряют свою актуальность и, если данные все еще остаются в хранилище данных, в течение длительных периодов времени накапливается все больше и больше неактуальных данных. Неправильные временные метки набора данных также приводят к тому, что информацию невозможно найти или оценить. И в таком случае образуется то, что называется болотом данных (data swamp).

Существуют типичные характеристики болота данных, на наличие которых вы можете проверить свое озеро данных и затем от них избавиться:

1. Большие данные без какой-либо организации и документации, например, через каталог данных или концепцию ролей.
2. Отсутствует метаинформация структурированных или неструктурированных данных.
3. Устаревшие и неверные данные.
4. Нет директора по данным или владельца продукта, который управляет платформой.
5. Отсутствующие или нарушенные связи между информацией.

Для очистки данных при замусоривании данных могут оказаться полезны такие роли, как владелец продукта или директор по цифровым технологиям, которые организуют и развивают Data lake. Кроме того, необходимо создать каталог данных, который обеспечит ясность данных. Вместе с концепцией ролей это гарантирует, что данные дойдут до нужных людей. Неверные и старые данные должны быть удалены или заархивированы, поскольку это в любом случае часто требуется нормативными актами и может также привести к снижению затрат. Требованиями к записи данных являются, например, маркировка источника данных, маркировка метаданных и содержательная номенклатура [18].

### Обзор платформ для создания и управления озёр данных

#### Google cloud

GCP предлагает набор услуг автоматического масштабирования, которые позволяют создать озеро данных, которое интегрируется с существующими приложениями. К ним относятся Dataflow и Cloud Data Fusion для поглощения данных, Cloud Storage для хранения, а также Dataproc и BigQuery для обработки данных и аналитики. Google Cloud предоставляет инструменты и рабочие процессы для управления озерами данных на протяжении всего их жизненного цикла. Google структурирует свои услуги озера данных по четырем ключевым этапам жизненного цикла озера данных [19]:

1. Приём — позволяет данным из многочисленных источников, таких как потоки данных о событиях, журналы и устройства IoT, хранилища исторических данных, данные из транзакционных приложений, поступать в озеро данных.
2. Хранение — хранение данных в надежном и легкодоступном формате.
3. Обработка — преобразование данных из исходного формата в формат, позволяющий использовать и анализировать.
4. Исследование и визуализация — анализ данных и представление их в виде визуализаций или отчетов, предоставляющих ценную информацию бизнес-пользователям.

Так же на этой платформе имеется возможность интеграции уже существующих Data lake из некоторых других платформ.

#### Hadoop Azure Data Lake

Является платформой, в которой и создавалась концепция Data lake [20]. В озерах данных данные чаще всего хранятся в распределенной файловой системе Hadoop (HDFS). Эта система позволяет осуществлять одновременную обработку данных. Это связано с тем, что при приеме данные разбиваются на сегменты и распределяются по разным узлам кластера.

HDFS обладает рядом отличительных свойств [21]:

1. Большой размер блока по сравнению с другими файловыми системами (>64MB), поскольку HDFS предназначена для хранения большого количества огромных (>10GB) файлов;
2. Ориентация на недорогие и, поэтому не самые надежные сервера – отказоустойчивость всего кластера обеспечивается за счет репликации данных;
3. Зеркалирование и репликация осуществляются на уровне кластера, а не на уровне узлов данных;
4. Репликация происходит в асинхронном режиме – информация распределяется по нескольким серверам прямо во время загрузки, поэтому выход из строя отдельных узлов данных не повлечет за собой полную пропажу данных;
5. HDFS оптимизирована для потоковых считываний файлов, поэтому применять ее для нерегулярных и произвольных считываний нецелесообразно;
6. Клиенты могут считывать и писать файлы HDFS напрямую через программный интерфейс Java;
7. Файлы пишутся однократно, что исключает внесение в них любых произвольных изменений;
8. Принцип WORM (Write-once and read-many, один раз записать – много раз прочитать) полностью освобождает систему от блокировок типа «запись-чтение». Запись в файл в одно время доступен только одному процессу, что исключает конфликты множественной записи.
9. HDFS оптимизирована под потоковую передачу данных;
10. Сжатие данных и рациональное использование дискового пространства позволило снизить нагрузку на каналы передачи данных, которые чаще всего являются узким местом в распределенных средах;
11. Самодиагностика — каждый узел данных через определенные интервалы времени отправляет диагностические сообщения узлу имен, который записывает логи операций над файлами в специальный журнал;
12. Все метаданные сервера имен хранятся в оперативной памяти.

#### AWS EMR

AWS EMR - это сервис, предоставляемый Amazon Web Services (AWS), который позволяет организациям хранить и анализировать большие объемы данных в облаке. Он объединяет в себе возможности сервиса EMR (Elastic MapReduce) для обработки и анализа больших данных с функциональностью Data Lake, обеспечивая пользователям удобный и масштабируемый способ работы с данными.

Особенность AWS EMR, что она использует все продукту от AWS. Для обработки используется AWS Lake Formation, его особенности [22]:

1. Импортирование данных из существующих баз данных. Данные сканируются, когда пользователь предоставляет AWS Lake Formation местоположение текущих баз данных и свои данные для входа.
2. Организация и маркировка данных. Lake Formation предлагает коллекцию технических метаданных, извлеченных из источников данных, для потребителей, которые ищут наборы данных.
3. Преобразование данных. Такие преобразования, как перезапись форматов дат для обеспечения единообразия, возможны с помощью Lake Formation. Amazon Data Lake Formation создает шаблоны преобразований и организует процессы, которые будут их выполнять.
4. Принудительное шифрование. Пользовательское Data lake зашифровано с помощью шифрования Amazon S3 через Lake Formation. Чтобы предотвратить удаление вредоносных данных при передаче, можно использовать отдельные учетные записи для исходного и целевого регионов при использовании S3.
5. Управление контролем доступа. Lake Formation позволяет управлять разрешениями на доступ к данным в Data lake из одного места. Доступ к данным можно ограничить на уровне базы данных, таблицы, столбца, строки и ячейки с помощью правил безопасности. Эти политики применяются к пользователям и ролям, а также к пользователям и группам, объединенным через внешнего поставщика удостоверений.
6. Настройте ведение журнала аудита. Мониторинг доступа к данным на платформах аналитики и машинного обучения.
7. Метатеги данных для бизнеса. В Data Lake на Amazon можно определить соответствующие варианты использования и уровни конфиденциальности данных, используя безопасность формирования и ограничения доступа.
8. Поиск данные для анализа. Пользователи Lake Formation имеют доступ к текстовому поиску, выполняемому онлайн, для поиска и фильтрации наборов данных, хранящихся в общей библиотеке данных.

#### Microsoft Azure Data Lake

В Azure Data Lake представлены все возможности, упрощающие хранение данных любых объема, формата и скорости передачи, а также выполнение любых видов обработки и анализа на разных платформах и языках для разработчиков, специалистов по обработке и анализу данных и аналитиков. Azure Data Lake упрощает получение и хранение данных, одновременно ускоряя работу пакетной, потоковой и интерактивной аналитики.

Особенность заключается в заранее собранном наборе инструментов, таких как [23]:

1. Azure HDInsight — это управляемая комплексная облачная служба аналитики с открытым кодом, предназначенная для предприятий. С помощью HDInsight можно использовать платформы с открытым кодом, такие как Apache Spark, Apache Hive, LLAP, Apache Kafka, Hadoop и т. д. в среде Azure. Azure HDInsight можно применять в различных сценариях обработки больших данных. Это могут быть исторические данные (данные, которые уже собираются и хранятся) или данные в режиме реального времени (данные, которые передаются непосредственно из источника).
2. Data Lake Store — это высокомасштабируемое облачное озеро данных, предназначенное для предприятий, создано в соответствии с открытыми стандартами HDFS. В нём отсутствии ограничений на размер данных, есть возможностью выполнять огромное количество параллельных аналитических задач и имеется единая платформа хранения данных. Так же имеется проверка подлинности данных с помощью Microsoft Entra ID и управления доступом на основе ролей.
3. Data Lake Analytics — служба заданий аналитики. Это облачная служба аналитики, в которой можно с легкостью разрабатывать и выполнять программы обработки и программы массовых параллельных операций преобразования данных на U-SQL, R, Python и .NET. В ней нет инфраструктуры, так как нет серверов, виртуальных машин или кластеров, которые нужно ждать, настраивать и которыми нужно управлять. Можно масштабировать вычислительную мощность, измеряемую в единицах Azure Data Lake Analytics (AU). Имеются специальные библиотеки на языке .NET, R и Python с помощью, которых обрабатываются данные.

### Определение критических различий

Исходя из написанного выше, можно выделить следующие значимые отличия от озера данных:

1. Данные не хранятся на сервере;
2. Возможность простого подключить БД уже имеющийся СУБД
3. Формат доступа к данным и возможность их редактирование с ограничениями используемых СУБД.

## Описание системы

### Описание взаимодействия модулей

В данной работе разрабатывается системы для взаимодействия пользователя с множеством баз данных, находящимися в различных системах управления базами данных. Эта система разработана с целью сделать процесс получения информации более интуитивным и удобным для пользователей. Первоначально пользователю необходимо выбрать нужные ему базы данных и соответствующие СУБД, из которых он планирует извлекать данные. Список доступных для выбора СУБД заранее определён разработчиком. Для извлечения данных нужно писать запросы по разработанному синтаксису. Пользователь имеет возможность получать информацию из нескольких баз данных и СУБД одновременно. Кроме того, данная система обладает гибкостью, так как имеет возможность использовать потенциально любую СУБД, при условии, что для этого предварительно добавлен необходимый функционал, соответствующий указанному шаблону.

Всего в системе можно выделить 3 части - ввод запроса пользователем, парсер запроса для каждой СУБД, исполнение запроса каждой СУБД. Визуально их взаимодействие показано на рис. 1, текстовое описание представлено далее.

Первая часть - ввод запроса пользователем. Пользователь в браузере вводит запрос на получение нужных ему данных в соответствии с специальном синтаксисом, строение которого описано далее.

Вторая часть - парсер запросов. В этом элементе системы запрос пользователя разбивается на части для каждой БД. Эти части запроса в нужной для корректного исполнения последовательности подаются в соответствующие классы определённой СУБД, после чего возвращают результат, который уже обрабатывается и выводится пользователю или добавляется в запрос для следующего класса СУБД, если это не конечный результат.

Третья часть, которую можно выделить, это - классы СУБД. Каждый такой класс представляет собой возможность взаимодействия с СУБД для выполнения запроса, а его объекты представляют каждую БД в этой СУБД. Каждый объект класса должен преобразовывать часть запроса, пришедшего к нему от парсера, в запрос на язык, который является командой для СУБД, к классу которой он относится. После преобразования он обращается с этой командой к БД, к которой он относится, и возвращает результат в парсер.

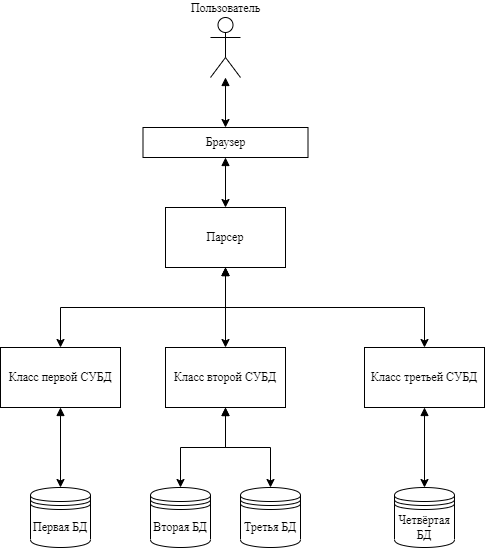


Рисунок 1. Взаимодействие между элементами

### Синтаксис запросов между разными СУБД

Чтобы была возможность делить запросы для исполнения на разных СУБД необходимо сделать свой синтаксис запросов. Далее представлено описание синтаксис основных запросов.

Запрос будет записываться одной строкой, в которой будут писаться название БД, путь к данным из БД и операторы с разделителями в виде точек.

Пример запроса:

dbcity.read(dbcity.building.address).where(dbpeople.read(dbpeople.personal\_data.name).where(dbpeople.personal\_data.id=1)=dbcity.building.address\_owner)

Сначала пишется название БД. После этого через разделитель - точку, пишется оператор, применяемый к указанной БД.

В системе будут представлены следующие операторы:

* x.y.z, где x – название БД, y – название сущности, являющийся аналогом таблицы из реляционных БД, из ранее обозначенной базы данных, z – название сущности, являющийся аналогом столбцов из реляционных БД, из ранее обозначенной таблицы. Таким образом из БД можно получить необходимые данные.
* where – оператор, в котором в скобках описывается фильтр, по которому выбираются данные из БД. В условии используются оператор доступа к данным и операторы сравнения. Сравниваемыми элементами могут являться данные из БД или константы. Условия могут вложенными. Так же возможно использование операторов AND и OR.
* create – вставка данных в БД. У оператора в скобках сначала через запятую указываются операторы доступа к данным, которые означают новые поля, после чего в конце через запятую указываются вставляемые данные в виде массива списков. Например, dbcity.create(dbcity.building.address, dbcity.building.owner, [[“Wall Street”, “Stan Smith”], [“Broadway”, “John Doe”]]).
* read – чтение данных из БД. У оператора в скобках указываются операторы доступа к данным, которые должны быть прочитаны. Например, dbcity.read(dbcity.building.address).
* update – изменение данных в БД. У оператора в скобках сначала через запятую указываются операторы доступа к данным, которые означают поля, в которых будут производится изменения, после чего в конце через запятую указываются вставляемые данные в виде списка. Например, dbcity.update(dbcity.building.address, dbcity.building.owner, [“Broadway”, “John Doe”])
* delete – удаление данных из БД. После данного оператора указывается оператор where. Например, dbcity.delete.where(dbcity.building.owner=“John Doe”).

### Классы СУБД

В системе для каждой СУБД создаётся собственный класс. Для каждой БД, которую подключает пользователь создаётся собственный объект соответствующего класса СУБД. Все классы наследуются от шаблона, пример для Neo4j, Cassandra и MongoDB представлен на рис. 2.

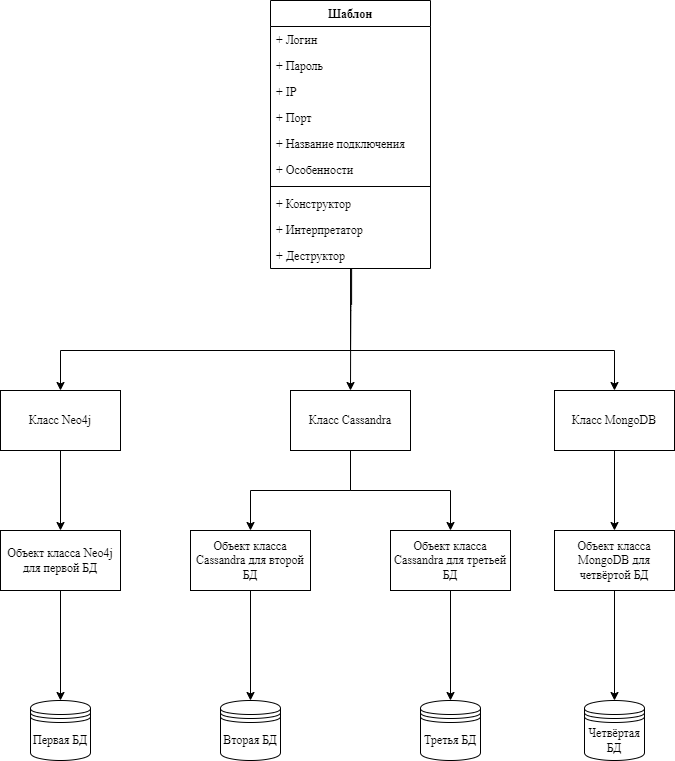


Рисунок 2. Схема работы классов СУБД

Сам шаблон должен иметь следующую структуру полей:

* Логин – логин, который нужен для подключения к базе данных.
* Пароль – пароль, который нужен для подключения к базе данных.
* IP – ip для подключения к базе данных.
* Порт – порт для подключения к базе данных.
* Название подключения - название, которое необходимо для различия между разными БД одной СУБД.
* Особенности – текстовая строка с необходимой информации для корректного выполнения запросов.

Так же шаблон должен содержать следующие методы:

* Конструктор. Метод, которому на вход подаётся - адрес базы данных; порт базы данных; логин для авторизации; пароль для авторизации; название базы данных. Данный метод создаёт и хранит подключение и название базы данных.
* Интерпретатор. Метод, которому на вход подаётся - запрос на синтаксисе, описанном в предыдущей главе. Данный метод преобразует полученный запрос в формат, нужный для конкретной СУБД и исполняет его. В конце возвращает массив кортежей с полученным результатом.
* Деструктор. Метод, который закрывает подключение для текущей БД.

### Парсер запросов

В описываемой в данной статье системе запрос от пользователя необходимо разделить на подзапросы. После этого разбитые подзапросы конвертируются в запросы, которые могут воспринимать СУБД, для которых эти запросы предназначены и исполняются.

Сначала запрос читается слева направо и разбивается на подзапросы. Далее подзапросы конвертируются в читаемый для СУБД вид и исполняются справа налево с передачей результата запроса. Данный процесс подробно изображен на рисунке 3.

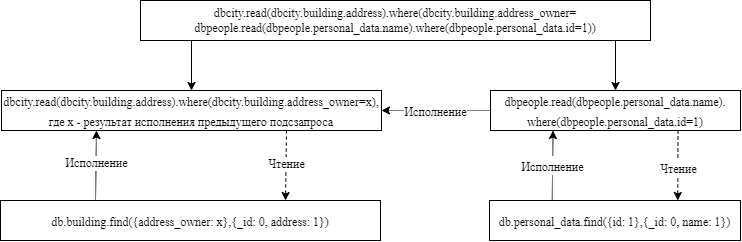


Рисунок 3. Деление основного запроса на подзапросы для каждой БД

Конвертацию подзапросов осуществляет общий для всех классов БД метод Интерпретатор. Рассмотрим работу одного из них, а именно интерпретатора класса MongoDB.

Был выделен следующий подзапрос:

dbpeople.read(dbpeople.personal\_data.name).where(dbpeople.personal\_data.id=1)

Сначала идёт подключение к базе данных dbpeople. Парсер начинает выполнять с более меньшего подзапроса:

(dbpeople.personal\_data.name).where(dbpeople.personal\_data.id=1), где dbpeople – БД в СУБД MongoDB.

Часть запроса dbpeople.personal\_data.name преобразуется в {\_id: 0, name: 1},where(dbpeople.personal\_data.id=1) преобразуется в {\_id: 1}.

Далее конвертируется часть запроса dbpeople.read() и исполняется в виде следующего запроса:

db.personal\_data.find({\_id: 1},{\_id: 0, name: 1}).

Для создания очереди на исполнения создаётся стек, со следующим строением:

1. Название подключения – какое подключение необходимо выполнить для исполнения запроса.
2. Команда – название команды, которая будет исполняться
3. Аргументы команды – аргументы, которые были указаны при вызове команды.
4. Where список – список, в котором хранятся символы, которые необходимо учитывать при анализе условия – “(”, ”)”, ”AND”, ”OR”, ”=”, “!=”, “<”, ”>”, “<=”, “>=”
5. Номер итерации – положение элемента в стеке, в который необходимо вставить результат
6. Позиция в where списке – в какой порядковый номер необходимо вставить результат в where списке

Сначала в запросе ищется первая из команд, после чего в элемент стека заносится подключение для которого была вызвана команда и затем сама команда. После ищется конец аргумента, начиная с первого символа, если количество открывающих скобок равно -1 (так как первая уже учтена) и встречается закрывающая, то это конце аргумента. Весь аргумент заносится в элемент стека. Далее проверяется имеется ли у команды оператор where. Если он отсутствует, то заносится пустой список и на место координат вставки результата записывается число -1. В случае наличия оператора where ищется конец его аргумента. И после система пытается найти есть ли в аргументе операторы AND или OR. Для этого ищутся координаты начала и конца первого названия путём посимвольного прохождения всей строки, пока не будет встречен символ. После этого всё что до первого аргумента вносится отдельно в список where и вместо аргумента ставится @. Данный процесс продолжается пока не закончится проверяемая строка. Все найденные аргументы заносятся в отдельный список. После для каждого аргумента в списке ищется разделитель аналогично поиску конца аргумента. Далее каждый аргумент разделителя, если они простые заносятся вместо @. Если же они сложные, то для них процесс начинается сначала, но уже заносят результат в итерацию и место, где оно стоит. Место помечается знаком %.

Последовательно из стека берётся запись, которая передаётся в соответствующий интерпретатор, который определяется по первому значению в этой записи. После выполнения запроса берётся результат и заносится вместо символа % в where список нужного элемента стека. Место для вставки указано последними 2 элементами в записи стека. И данный процесс продолжается пока не закончатся записи в стеке.

# Возможности применения в других областях

В настоящее время наблюдается продолжающийся рост систем, поддерживающих огромный объем реляционных и нереляционных форм данных. Наличие единой системы данных для управления, как хорошо структурированными данными, так и данными NoSQL выгодно пользователям, поскольку для каждой конкретной задачи существуют более предпочтительные варианты хранения данных в зависимости от типа СУБД, таким образом, предоставление разных структур хранения данных в одной системе позволяет сделать информацию более доступной и понятной для пользователей. Такая система также улучшает визуализацию и понимание данных.

Проблема взаимодействия между различными СУБД и БД существует уже давно [24, 25, 26]. Попытка создания аналогичной системы уже были, однако она была представлена для специализированной сферы, что представлено в статье [27], также уже была попытка создать систему связывающую различные БД через JSON запросы [28], а рассматриваемая в данной работе система также является универсальной, что выражается в поддержке нескольких СУБД и возможности подключать множество баз данных и брать информацию из нескольких источников одновременно, однако работает по другому принципу. В описываемой системе не производится сбор данных из разных БД, а идёт обращение непосредственно к СУБД.

В биологии специалистам регулярно приходится работать с информацией из различных баз данных и систем управления ими (СУБД). Например, исследования РНК требуют одновременного анализа данных из разных источников, что значительно усложняет работу ученых [29, 30]. Предлагаемая технология может помочь в решении этой проблемы.

В политологии также возникает необходимость в интеграции данных. На сегодняшний день существует множество баз, содержащих систематизированную информацию по административно-территориальным единицам (АТЕ) субнационального уровня, включая регионы. Эти данные собираются и публикуются государственными статистическими службами, однако методы их обработки и форматы представления могут сильно различаться в зависимости от страны. Для государств-членов ЕС социально-экономические показатели регионов унифицированы в базе Евростата. Что касается стран, не входящих в Евросоюз, аналогичные данные можно найти в базах Организации экономического сотрудничества и развития (ОЭСР) [31].

Кроме того, крупные компании сталкиваются с аналогичными сложностями при взаимодействии между своими подразделениями, каждое из которых использует собственную СУБД и работает обособленно. В таких условиях интеграция данных становится ключевой задачей.

# Заключение

В данной работе для обеспечения эффективного взаимодействия между различными системами управления базами данных (СУБД) были тщательно изучены и проанализированы синтаксисы запросов, архитектурные особенности и функциональные встроенных в систему СУБД. Особое внимание уделялось сравнению их языков запросов. Для всестороннего анализа преимуществ и недостатков современных систем хранения и обработки данных было проведено исследование различных подходов к организации озер данных.

В результате проведенного анализа в данной работе предложена унифицированная система, которая предоставляет пользователю удобный и интуитивно понятный интерфейс для доступа к данным из различных СУБД. Система позволяет значительно упростить процесс работы с разнородными источниками данных, сократить время на их обработку и предоставить конечным пользователям более понятный и структурированный доступ к информации.

# Список источников

1. The Multi-model Databases – A Review / Ewa Płuciennik & Kamil Zgorzałek; — Springer, Cham: Communications in Computer and Information Science, vol. 716, 2017. — 141–152 с.
2. Key-Value stores:a practical overview / Marc Seeger; — Stuttgart, Germany, 2009.
3. Survey of graph database models / Renzo Angles, Claudio Gutierrez; — ACM Computing Surveys, vol. 40, 2008. — 1–39 с.
4. An Approach for Implementing Online Analytical Processing Systems under ColumnFamily Databases / Abdelhak Khalil and Mustapha Belaissaoui; — IAENG International Journal of Applied Mathematics, vol. 53, 2023. — 31–39 c.
5. What is a Document Database? // mongodb.com – URL: https://www.mongodb.com/document-databases (дата обращения: 23.04.2025).
6. Multi-model database // wikipedia.org – URL: https://en.wikipedia.org/wiki/Multi-model\_database (дата обращения: 23.04.2025).
7. Мультимодельные СУБД — основа современных информационных систем? // habr.com – URL: https://habr.com/ru/articles/462493/ (дата обращения: 23.04.2025).
8. Polyglot database architectures = polyglot challenges / Lena Wiese; — Gottingen, Germany: CEUR Workshop Proceedings, vol. 1458, 2015. — 422-426 с.
9. MongoDB: The Definitive Guide / Shannon Bradshaw, Eoin Brazil and Kristina Chodorow. — Boston: O'Reilly Media, Inc., 2019. — 511 с.
10. Query Documents // mongodb.com – URL: https://www.mongodb.com/docs/manual/tutorial/query-documents/ (дата обращения: 23.04.2025).
11. What is Neo4j? // neo4j.com – URL: https://neo4j.com/docs/getting-started/whats-neo4j (дата обращения: 23.04.2025).
12. What is Apache Cassandra? // cassandra.apache.org – URL: https://cassandra.apache.org/\_/index.html (дата обращения: 23.04.2025).
13. Data lake // en.wikipedia.org – URL: https://en.wikipedia.org/wiki/Data\_lake (дата обращения: 23.04.2025).
14. Чем озеро данных отличается от базы и зачем оно нужно аналитикам // practicum.yandex.ru – URL: https://practicum.yandex.ru/blog/chto-takoe-ozera-dannyh (дата обращения: 23.04.2025).
15. Что такое озера данных и почему в них дешевле хранить big data // cloud.vk.com – URL: https://cloud.vk.com/blog/chto-takoe-ozera-dannyh-i-zachem-tam-hranyat-big-data (дата обращения: 23.04.2025).
16. Pwint Phyu Khine, Zhao Shun Wang Data Lake: A New Ideology in Big Data Era. – Wuhan, China, 2017
17. Data Lake // yandex.cloud.ru – URL: https://yandex.cloud.ru/docs/glossary/datalake (дата обращения: 23.04.2025).
18. What is a Data Swamp? // medium.com – URL: https://medium.com/codex/what-is-a-data-swamp-38b1aed54dc6 (дата обращения: 23.04.2025).
19. Google Cloud Data Lake: 4 Phases of the Data Lake Lifecycle // bluexp.netapp.com – URL: https://bluexp.netapp.com/blog/gcp-cvo-blg-google-cloud-data-lake-4-phases-of-the-data-lake-lifecycle (дата обращения: 23.04.2025).
20. Pentaho, Hadoop, and Data Lakes // jamesdixon.wordpress.com – URL: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/ (дата обращения: 23.04.2025).
21. HDFS // bigdataschool.ru – URL: https://bigdataschool.ru/wiki/hdfs (дата обращения: 23.04.2025).
22. AWS Lake Formation: Overview, Architecture & Functionality // k21academy.com – URL: https://k21academy.com/amazon-web-services/aws-data/aws-lake-formation/ (дата обращения: 23.04.2025).
23. Azure Data Lake // azure.microsoft.com – URL: https://azure.microsoft.com/ru-ru/solutions/data-lake (дата обращения: 23.04.2025).
24. When will we have true heterogeneous database systems / A. P. Sheth; — In Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow (ACM '87). — IEEE Computer Society Press, Washington, DC, USA, 1987. — 747–748 c.
25. Principles of Distributed Data Management in 2020? / Patrick Valduriez; — INRIA and LIRMM, Montpellier – France, 2020.
26. Blinova O.V., Pankratova E. V., Farkhadov M.P. Principles of construction and analysis of architectures for modern scientific information systems // 7th International Scientific Conference. 2023. P. 111-113. DOI 10.25728/icct.2024.032.
27. Haas L.M., Rice J.E., Schwarz P.M., Swope W.C. DISCOVERYLINK: a system for integrated access to life sciences data sources // IBM systems journal, Vol. 40, №2, P. 489-511, 2001. DOI 10.1147/sj.402.0489.
28. Zhang L., Pang Ke., Xu J., Niu B. JSON-based control model for SQL and NoSQL data conversion in hybrid cloud database // Journal of cloud computing, Vol. 11, №1, P. 1-12, 2022. DOI 10.1186/s13677-022-00302-9.
29. Schäfer R.A., Rabsch D., Scholz G.E., Stadler P.F., Hess W.R., Backofen R., Fallmann J., Voß B. RNA interaction format: a general data format for RNA interactions // Bioinformatics, Vol. 39, №11, P. 1-3, 2023. DOI 10.1093/bioinformatics/btad665.
30. Martha V.S, Liu ZH., Guo Li., Su Zh., Ye Ya., Fang H., Ding D., Tong W., Xu X. Constructing a robust protein-protein interaction network by integrating multiple public databases // Bioinformatics, Vol. 12, №10, P. 1-10, 2011. DOI 10.1186/1471-2105-12-s10-s7.
31. Панов П.В. База данных «субнациональный регионализм и многоуровневая политика (reg-mlg)» // Вестник Пермского университета. ПОЛИТОЛОГИЯ, Т. 15, №4, С 111-120, 2021. DOI 10.17072/2218-1067-2021-4-111-120.