|  |  |
| --- | --- |
| 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: «Искусственный интеллект в задачах бизнес-аналитики»

Студент Журавлев Николай Вадимович

Группа ИУ5-44М

Название Анализ методов 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](#_Toc196906510)

[Что такое межсетевого взаимодействия между различными СУБД 3](#_Toc196906511)

[Суть технологии 3](#_Toc196906512)

[Ожидаемые преимущества 4](#_Toc196906513)

[Виды СУБД для взаимодействия между ними 4](#_Toc196906514)

[Сравнение с озером данных 5](#_Toc196906515)

[Группы методов взаимодействия между различными СУБД 9](#_Toc196906516)

[Специально разработанная система для конкретных случаев 9](#_Toc196906517)

[DiscoveryLink 9](#_Toc196906518)

[MIKS 12](#_Toc196906519)

[hStorage-DB 13](#_Toc196906520)

[Интеграция СУБД 15](#_Toc196906521)

[Создание нового СУБД 19](#_Toc196906522)

[Сохранение данные в облако 21](#_Toc196906523)

[Унифицировать СУБД 22](#_Toc196906524)

[Модификация UDBMS 22](#_Toc196906525)

[Встроить на уровне системы 27](#_Toc196906526)

[Графовое взаимодействие 27](#_Toc196906527)

[Получение данных на основе JSON 30](#_Toc196906528)

[Объём рынка 32](#_Toc196906529)

[Области применения 32](#_Toc196906530)

[Заключение 33](#_Toc196906531)

[Список источников 33](#_Toc196906532)

# Введение

Современные системы демонстрируют устойчивый рост, поддерживая как реляционные, так и нереляционные форматы данных. Многомодельные базы данных работают с различными типами моделей, включая документные, графовые, реляционные и ключ-значение [1]. Использование единой системы для управления структурированными и NoSQL-данными дает пользователям преимущество: в зависимости от задачи можно выбрать оптимальный способ хранения информации. Объединение разных структур данных в одной системе повышает доступность и удобство их восприятия, а также улучшает визуализацию и анализ данных.

# Что такое межсетевого взаимодействия между различными СУБД

В этой работе детально исследуются системы, обеспечивающие взаимодействие пользователя с множеством баз данных, работающих под управлением различных СУБД.

Пользователь сначала выбирает базы данных и соответствующие СУБД, из которых требуется извлечь информацию. Системы позволяет получать данные из нескольких источников одновременно, обеспечивая гибкость за счёт поддержки потенциально любых СУБД — при условии предварительной интеграции необходимого функционала в соответствии с предложенным шаблоном.

# Суть технологии

Необходима система, обеспечивающая взаимодействие пользователя с множеством баз данных, расположенных в различных системах управления базами данных (СУБД).

Пользователь сначала выбирает необходимые базы данных и соответствующие СУБД, из которых требуется извлекать информацию. Перечень доступных СУБД заранее задан разработчиком. Система позволяет выполнять запросы к нескольким базам данных и СУБД одновременно.

# Ожидаемые преимущества

Данная система создана для того, чтобы упростить и сделать более интуитивным процесс работы с информацией, что особенно важно в условиях постоянного роста объёмов данных и многообразия их источников. Использование единой платформы для управления как структурированными, так и NoSQL-данными даёт пользователям преимущество: в зависимости от конкретной задачи можно выбрать оптимальный способ хранения информации, поскольку разные типы СУБД лучше подходят для разных типов данных. Благодаря поддержке различных структур хранения в одной системе, данные становятся более доступными и легче воспринимаются.

# Виды СУБД для взаимодействия между ними

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 [2].

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 [3].

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 [4].

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 [5].

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 [6].

Так же необходимо учесть существование мультимодельных баз данных. 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 [7].

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» [8].

# 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) [9].

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 [10].

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 [11].

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 [11, 12].

Озера данных оптимизированы для высокой пропускной способности, но ради этого приходится жертвовать качеством данных [13]:

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 [14].

# Группы методов взаимодействия между различными СУБД

Проблема взаимодействия между различными СУБД и БД существует уже давно [15, 16]. Далее представлены некоторые способы её решения, поделённые на группы.

## Специально разработанная система для конкретных случаев

В данной группе системы специально создаются системы, которые предназначены для решения проблемы в конкретной среде. Примеры таких систем представлены далее.

### DiscoveryLink

Общая архитектура DiscoveryLink является общей для многих гетерогенных систем баз данных, включая TSIMMIS, DISCO, Pegasus, DIOM, HERMES и Garlic. Приложения подключаются к серверу DiscoveryLink с помощью любого из множества стандартных клиентских интерфейсов базы данных, таких как OpenDatabase Connectivity (ODBC) или Java DatabaseConnectivity (JDBC\*\*), и отправляют запросы в DiscoveryLink в стандартном SQL. Информация, необходимая для ответа на запрос, поступает из одного или нескольких источников данных, которые были идентифицированы в DiscoveryLink через процесс, называемый регистрацией. Источники данных, представляющие интерес для наук о жизни, варьируются от простых файлов данных до сложных доменно-специфичных систем, которые не только хранят данные, но и включают специализированные алгоритмы для поиска или обработки данных. Возможность использования этих специализированных возможностей не должна быть утеряна при доступе к данным через DiscoveryLink.

Когда приложение отправляет запрос на сервер DiscoveryLink, сервер определяет соответствующие источники данных и разрабатывает план выполнения запроса для получения запрошенных данных. План обычно разбивает исходный запрос на фрагменты, которые представляют работу, которая должна быть делегирована отдельным источникам данных, плюс дополнительную обработку, которая должна быть выполнена сервером DiscoveryLink для дальнейшей фильтрации, агрегации или слияния данных. Способность сервера DiscoveryLink дополнительно обрабатывать данные, полученные из источников, позволяет приложениям использовать всю мощь языка SQL, даже если часть запрашиваемой ими информации поступает из источников данных с небольшими или отсутствующими собственными возможностями обработки запросов, такими как файлы. Сервер DiscoveryLink взаимодействует с источником данных с помощью оболочки, программного модуля, адаптированного для определенного семейства источников данных. Оболочка для источника данных отвечает за четыре задачи:

1. Отображение информации, хранящейся в источнике данных, в реляционную модель данных DiscoveryLink
2. Информирование DiscoveryLink о возможностях обработки запросов источников данных
3. Отображение фрагментов запроса, отправленных в оболочку, в запросы, которые могут быть обработаны с использованием собственного языка запросов или программного интерфейса источника данных
4. Выдача таких запросов и возврат результатов после их выполнения

Поскольку оболочки являются ключом к расширяемости в DiscoveryLink, одной из основных целей для архитектуры wrap-per было обеспечение реализации оболочек для максимально широкого спектра источников данных с минимальными усилиями. Чтобы сделать диапазон источников данных, к которым можно получить доступ с помощью DiscoveryLink, максимально широким, требуется только, чтобы источник данных (или приложение) имел некоторую форму программного интерфейса, который может отвечать на запросы и, как минимум, мог возвращать неотфильтрованные данные, смоделированные как строки таблицы. Автору оболочки не нужно реализовывать стандартный интерфейс запроса, который может быть слишком высокоуровневым или слишком низкоуровневым для базового источника данных. Вместо этого оболочка предоставляет информацию о возможностях обработки запросов источника данных и специализированных средствах поиска серверу DiscoveryLink, который динамически определяет, какую часть данного запроса источник данных способен обработать. Этот подход позволяет быстро создавать оболочки для простых источников данных, сохраняя при этом возможность использовать уникальные возможности обработки запросов нетрадиционных источников данных, таких как поисковые системы для химических структур или изображений. Оболочка может быть написана с минимальным знанием внутренней структуры DiscoveryLink. В результате стоимость написания базовой оболочки невелика. Оболочка, которая просто делает данные из нового источника доступными для DiscoveryLink, не пытаясь использовать большую часть собственных возможностей обработки запросов источника данных, может быть написана за считанные дни. Поскольку сервер DiscoveryLink может компенсировать отсутствующую функциональность в источниках данных, даже этот вид простой оболочки позволяет приложениям применять всю мощь SQL для извлечения новых данных и интеграции данных с информацией из других источников, хотя, возможно, и с производительностью ниже оптимальной. После написания базовой оболочки ее можно постепенно улучшать, чтобы использовать больше возможностей обработки запросов источника данных, что приводит к повышению производительности и повышению функциональности, поскольку раскрываются специализированные алгоритмы поиска или другие новые возможности обработки запросов источника данных. Оболочка DiscoveryLink — это программа на языке C++, упакованная как общая библиотека, которая может динамически загружаться сервером DiscoveryLink при необходимости. Обычно одна оболочка способна получать доступ к нескольким источникам данных, если они используют общий или похожий интерфейс прикладного программирования (API). Это происходит потому, что оболочка не кодирует информацию о схеме, используемой в источнике данных. Таким образом, схемы могут развиваться без необходимости внесения изменений в оболочку, пока API источника остается неизменным. Например, оболочка Oracle, предоставляемая DiscoveryLink, может использоваться для доступа к любому количеству баз данных Oracle, каждая из которых имеет свою схему. Фактически, та же оболочка поддерживает несколько уровней релиза Oracle [17].

### MIKS

Как и в других системах MIKS решили придерживаться «семантического подхода» к интеграции информации, т. е. представить содержимое источников информации с помощью концептуальных схем (или, другими словами, метаданных). Это позволяет обрабатывать данные не только с синтаксической точки зрения, но и в соответствии с их значением, чтобы выводить экстенсиональные и интенсиональные отношения между ними.

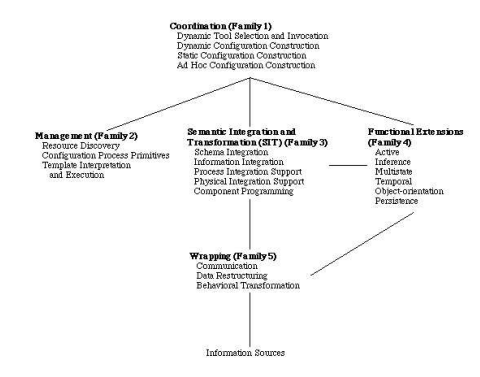


Рисунок 1. Архитектура MIKS

На рисунке 1 показаны пять основных семейств служб I^3 и основные способы их взаимодействия. В частности, можно определить две основные оси, чтобы подчеркнуть различные роли служб I^3. Вертикальная ось, которая охватывает семейства (1), (3), (5), сосредоточена на потоке и манипулировании информацией от необработанных источников информации до служб координации. Затем горизонтальная ось соединяет семейство управления и семейство координации. Эта ось подчеркивает критический аспект архитектуры ссылок I^3, то есть роль семейства управления для поиска полезных источников информации и использования локальных структур данных. Система в основном использует сервисы, принадлежащие вертикальной оси: в частности, семейство (3) было полностью исследовано [18].

### hStorage-DB

Рисунок 2 показывает архитектуру hStorage-DB. Когда менеджер буферного пула отправляет запрос менеджеру хранилища, также передается связанная семантическая информация. hStorage-DB расширяем менеджер хранилища «таблицей назначения политик», которая хранит правила для назначения каждому запросу надлежащей политики QoS в соответствии с его семантической информацией. Политика QoS встроена в исходный запрос ввода-вывода и доставляется в систему хранения через блочный интерфейс. hStorage-DB с использованием протокола Differentiated Storage Services от Intel Labs для доставки запроса и связанной с ним политики в гибридную систему хранения.

После получения запроса система хранения сначала извлекает политику и вызывает механизм для обслуживания этого запроса.

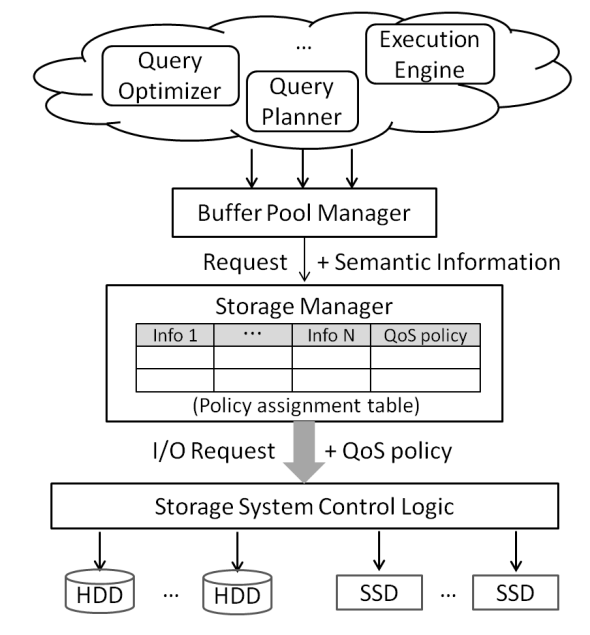


Рисунок 2. Архитектура hStorage-DB.

hStorage-DB основана на PostgreSQL 9.0.4. В основном это касается трех вопросов: (1) Она оснащена оптимизатором запросов и механизмом выполнения для извлечения семантической информации, встроенной в деревья планов запросов и в запросы пула буферов. (2) Имеет расширенную структуру данных пула буферов для хранения собранной семантической информации. Менеджер хранилища также был расширен для включения «таблицы назначения политик». (3) Наконец, поскольку PostgreSQL является многопроцессорной СУБД, для работы с параллелизмом был выделен небольшой регион общей памяти для глобальных структур данных, к которым должны иметь доступ все процессы.

Ключом к эффективности hStorage-DB является связывание каждого запроса с надлежащей политикой QoS [19].

## Интеграция СУБД

Данная группа пытается объединить несколько разных СУБД. Чем решает поставленную проблему.

Первый подход. Первым и обязательным условием является выбор СУБД для проектируемой гетерогенной системы баз данных, которая будет служить для исследований. Этот выбор показывает решающее влияние на последующие процессы разработки и оценки систем для объединения данных SQL и NoSQL. В качестве СУБД SQL был выбран PostgreSQL. PostgreSQL — это объектно-реляционная система на основе данных с открытым исходным кодом, которая использует расширяемый язык SQL в сочетании с некоторыми функциями, которые позволяют безопасно хранить и масштабировать сложные рабочие данные. К основным преимуществам PostgreSQL можно отнести высокую степень развития: в этой СУБД есть поддержка главных объектов и их поведений, включая типы данных, операции, функции, индексы и домены. По этой причине PostgreSQL можно назвать действительно гибким. Кроме того, эта СУБД предоставляет возможность создавать, хранить и использовать сложные структуры данных. Также стоит отметить, что PostgreSQL поддерживает вложенные и составные конструкции, которые не применяются и основаны на существующих стандартных реляционных базах данных. В качестве нереляционной СУБД была выбрана MongoDB — система управления базами данных NoSQL, которая набирает всю большую популярность на рынке и выделяет среди конкурентов своей технологией масштабировать потребление. К преимуществам MongoDB можно отнести гибкость, масштабируемость, доступность и высокую производительность.

При проектировании базы данных для управления проектами необходимо учитывать множество аспектов, связанных с проектами управления. Следует учитывать, что разрабатываемая система может использовать команды, которые могут входить в состав различных отдельных компаний, а также просто обычных людей, некоторые из которых разделились в команде для разработки продукта.

В проектах часто могут использоваться различные факторы, основанные на различных типах баз данных. Причин для такого подключения может быть несколько: добавление новых баз данных для балансировки нагрузки, обеспечение хранения данных или использование ресурсов.

Для доступа к внешним данным (из одной базы данных в другую) используются оболочки (обертки) внешних данных (Foreign Data Wrappers). Обертка внешних данных — это библиотека, предназначенная для взаимодействия с внешним источником и загрузки данных из него. В качестве внешних источников могут выступать репозитории NoSQL или сторонние серверы Postgres.

В настоящее время доступно множество оберток внешних данных (FDW), которые позволяют серверу PostgreSQL работать с различными удаленными хранилищами данных.

Среди множества оберток внешних данных для баз данных NoSQL можно также найти FDW для MongoDB. Обертка данных MongoDB выполняет функцию соединения между сервером MongoDB и PostgreSQL, транслируя операторы PostgreSQL в запросы, понятные базе данных MongoDB. Для этого соединения поддерживаются операторы SELECT, INSERT, DELETE и UPDATE.

Чтобы настроить соединение между PostgreSQL и MongoDB для отправки запросов, вам потребуется установить расширение mongo\_fdw. Для компиляции mongo\_fdw требуются следующие библиотеки:

– libbson;

– libmongoc;

– json-c.

Библиотеки libbson и libmongoc необходимы для корректной работы mongo\_fdw, поскольку это расширение использует для работы драйвер языка C [20].

Второй подход – разработка интегрированной схемы. Этапы разработки интегрированной схемы:

* Предварительная интеграция, где входные схемы преобразуются, чтобы сделать их более однородными (как синтаксически, так и семантически); однородными (как синтаксически, так и семантически);
* Идентификация соответствия, посвященная идентификации и описанию межсхемных отношений;
* Интеграция, заключительный этап, который разрешает межсхемные конфликты и объединяет соответствующие элементы в интегрированную схему.

Предварительный этап интеграции. Установление общего понимания существующих данных является предпосылкой успешной интеграции баз данных. Для этой цели входные схемы обычно преобразуются, чтобы сделать их максимально однородными. Исследователи в области интеграции баз данных обычно предполагают, что все входные схемы выражены в одной и той же модели данных, так называемой «общей» модели данных (CDM). Этап перевода становится предпосылкой интеграции и рассматривается как отдельная проблема.

К сожалению, современное состояние перевода моделей данных не располагает инструментами для автоматического перевода. Текущие разработки сосредоточены на переводах между объектно-ориентированными и реляционными моделями.

Большинство исследователей отдают предпочтение объектно-ориентированной модели. Аргумент заключается в том, что она содержит все концепции других моделей и что можно использовать методы для реализации определенных правил отображения. Но чем богаче модель, тем сложнее будет процесс интеграции, так как возникнет много несоответствий из-за разных выборов моделирования разными дизайнерами. Для упрощения интеграции альтернативой является CDM с минимальной семантикой, где представления данных сводятся к элементарным фактам, для которых нет альтернативы моделирования, как в моделях с бинарными отношениями. Вместо того, чтобы иметь дело с конкретными переводами, например, из модели X в модель Y, в последних работах ищутся общие методы. Общие трансляторы используют метамодель, т. е. модель данных, способную описывать модели данных, для получения знаний о моделях данных. Затем переводы выполняются как последовательность процесса реструктуризации данных в базе данных метамодели (например, вложение плоских кортежей для получения вложенного кортежа). Модели данных не могут выразить всю семантику реального мира. Неполнота их спецификаций приводит к неоднозначностям в интерпретации схемы. Семантическое обогащение — это процесс получения дополнительной информации для разрешения таких неоднозначностей. Самый сложный случай — когда данные находятся в файлах, но понимание ненормализованных и плохо документированных реляционных баз данных также представляет собой серьезную проблему.

Шаг идентификации соответствия. Следующий шаг — это выявление общих черт. Базы данных содержат представления фактов реального мира (объектов, связей или свойств). Интеграция баз данных выходит за рамки представлений, чтобы сначала искать то, что представлено, а не то, как оно представлено. Поэтому говорят, что две базы данных имеют что-то общее, если факты реального мира, которые они представляют, имеют некоторые общие элементы или иным образом взаимосвязаны. Примером последнего является: две отдельные библиотеки, одна из которых специализируется на научных дисциплинах, а другая — на социальных науках, желающие создать интегрированную базу данных.

Было бы целесообразно создать интегрированные типы объектов, такие как Автор (соответственно Статья), представляющие всех авторов (соответственно все статьи) в любой библиотеке. Мы говорим, что два элемента (вхождение, значение, кортеж, ...) из двух баз данных соответствуют друг другу, если они описывают один и тот же факт реального мира (объект, связь или свойство). Вместо того, чтобы определять соответствия экстенсионально, между экземплярами, соответствия определяются интенционально, между типами. Пример: каждая статья S2 соответствует статье S1, так что значение заголовка в S1 равно значению заголовка в S2. Интенсиональное определение называется утверждением соответствия между схемами (ICA). Процесс интеграции состоит в определении этих ICA и предоставлении для каждого из них федеративным пользователям глобального описания со всеми доступными данными по связанным элементам. Федеративная система хранит глобальное описание в интегрированной схеме (IS) и определение отображений между IS и локальными схемами. Полная интеграция существующих баз данных требует исчерпывающей идентификации и обработки всех соответствующих ICA. Тем не менее, возможно принять инкрементальный подход, при котором IS плавно обогащается по мере постепенного выявления новых соответствий [21].

## Создание нового СУБД

В последние годы использование СУБД переместилось из традиционной области коммерческих приложений, таких как управление запасами и банковское дело, во многие новые области, такие как инженерия (например, CAD/CAM) и офисные приложения. В результате были обнаружены новые требования к СУБД, и было обнаружено, что текущие СУБД не поддерживают эти новые приложения.

Проект Advanced Information Management (AlM) в IBM Heidelberg ScientificCenter, созданный несколько лет назад в ожидании некоторых из этих передовых приложений, изначально видел необходимость включения текстового поиска в обычные средства управления данными в СУБД. Потребность в такой возможности была не только основана для поиска офисных документов, но также была обнаружена в инженерных приложениях, таких как we11. С тех пор были обнаружены другие основные возможности, необходимые для поддержки расходящихся приложений. Первоначальная мысль авторов статьи [22] состояла в том, чтобы построить систему поверх существующей СУБД. Анализ вскоре показал, что это нехороший подход, поскольку ограничения для вписывания в данную систему слишком строги, изменения, необходимые для достижения наших целей, слишком велики, и что полученная система, скорее всего, будет работать неудовлетворительно. Эти наблюдения и анализы привели их к решению, что необходим совершенно новый дизайн со всеми устраненными ограничениями.

В их статье описывается проектирование такой системы для достижения нужных им целей.

Одной из целей проектирования системы является поддержка неатомарных типов данных и отношений со сложными — т. е. неплоскими или ненормализованными — кортежами в дополнение к нормализованным отношениям и простым типам данных. Недавние применения СУБД показали, что ненормализованная форма отношений (также называемая иерархической структурой) может быть использована с выгодой.

Некоторые исследователи считают, что иерархической структуры, построенной с нормализованными отношениями, будет достаточно. Однако авторы считают, что есть значительные преимущества в реализации системы, которая напрямую поддерживает нормализованные и ненормализованные отношения одновременно. Часть привлекательности нормализованных отношений заключается в их математической обоснованности и трактуемости. Ненормализованная форма имеет схожую привлекательность, поскольку недавние теоретические работы показали, что алгебра для ненормализованных отношений, которая включает нормализованную алгебру отношений как частный случай, теперь доступна. Это даёт математическую обоснованность в качестве основы для построения такой системы.

Тогда вопрос заключается в том, как построить систему, которая будет иметь гибкость и эффективность, аналогичные системам отношений, основанным на нормализованной форме. Другая цель — иметь систему базы данных для поддержки исторических данных. Основная цель заключается в том, чтобы включить информацию о времени или данные истории в базы данных системы.

## Сохранение данные в облако

С обещанием недорогого доступа к гибким и эластичным вычислительным ресурсам предприятия все чаще переносят свои существующие рабочие нагрузки в облачные среды. Однако неоднородность и сложность устаревшей ИТ-инфраструктуры затрудняют оптимизацию процессов миграции в масштабах предприятия. В этой статье мы представляем Cloud Migration Orchestrator (CMO), фреймворк для автоматизации и координации крупномасштабной облачной миграции на основе технологии IBM Business Process Management (BPM) с предварительной аналитикой миграции.

CMO легко автоматизирует сложные и подверженные ошибкам задачи, охватывающие анализ локального центра обработки данных с использованием корреляций между появлениями компонентов промежуточного программного обеспечения и параллельное выполнение миграции за счет интеграции различных инструментов миграции поставщиков. CMO предлагает возможность самообслуживания с выполнением миграции «одним щелчком» и предоставляет решение для сохранения IP-адресов для дальнейшей минимизации усилий по исправлению рабочей нагрузки.

В статье [23] авторы представляют таксономию сетевых проблем, основанную на опыте миграции устаревших сред, и обсуждают, как автоматизировать и оптимизировать конфигурации сети. Для каждого шага процесса миграции, начиная с оценки перед миграцией и заканчивая конфигурацией после миграции и обсуждают эффективность возможностей миграции, включая четырехкратное улучшение процесса (по сравнению с традиционными подходами) с использованием автоматизации и оркестровки.

## Унифицировать СУБД

Можно каким-либо образом унифицировать модели БД и тогда с ними можно будет удобно взаимодействовать.

### Модификация UDBMS

Фундаментальную модель унифицированной системы управления данными можно проследить до системы управления объектно-реляционными базами данных (ORDBMS), которая интегрирует объектно-ориентированные функции в реляционную модель. Система ORDBMS может управлять различными типами данных, такими как реляционные, объектные, текстовые и пространственные, подключая доменно-специфические типы данных, функции и реализации индексов в ядра СУБД. Например, PostgreSQL поддерживает реляционные, пространственные и XML-данные. Oracle продолжает усилия OR по поддержке XML, JSON и графовых данных.

В чистой реляционной модели столбец таблицы должен быть встроенным скалярным типом. Таким образом, чистая модель RDB представляет собой набор элементов, где каждый элемент имеет встроенные скалярные типы. С другой стороны, модель объектной базы данных в ORDBMS по-прежнему имеет концепцию набора верхнего уровня (которая обычно известна как коллекция объектов). Коллекция объектов представляет собой набор, содержащий элементы произвольного сложного объекта. Сам объект представляет собой еще один набор своих элементов, каждый из которых может быть другим набором. Однако текущая проблема в управлении многомодельными данными заключается в том, что каждый объект, такой как XML, JSON и граф, моделирует свои собственные данные домена и имеет определенный язык запросов (например, SQL для реляционных данных, XQuery для XML и SPARQL для RDF). Таким образом, унифицированная многомодельная база данных должна предоставлять новую (логически) унифицированную модель данных, которая действует как глобальное представление для различных типов данных.

Такая абстракция может скрыть детали реализации данных от пользователей и облегчить глобальный доступ и запрос для различных типов данных. Текущие усилия авторов статьи [24] в этом направлении направлены на объединение пяти типов данных, включая отношение, ключ-значение, JSON, XML и граф. Эта цель может быть достигнута в два этапа. На первом этапе представляем гибкий способ представления графа, JSON, XML и моделей ключ-значение в виде унифицированной модели NoSQL (UNM) логически. На втором этапе будут исследованы новые подходы для объединения UNM и моделей отношений. В конечном итоге унифицированная модель может поддерживать все пять типов данных.

Эта модель определит глобальные представления и операции для пяти типов данных. Эта унифицированная модель данных заложит общие основы для доступа к многомодельным данным и управления ими. Гибкое управление схемами. Оригинальная ORDBMS предполагает идеальный мир, основанный на схемах. Полуструктурированные данные и неструктурированные данные бросают вызов ORDBMS с дизайном без схем.

Улучшение обнаружения схемы для всех видов данных является проблемой, и это еще один интерфейс, которого не хватает в исходной ORDBMS. В ORDBMS нет индексирующего интерфейса обнаружения схемы.

Эволюция модели. С ростом зрелости баз данных NoSQL многие приложения переходят на хранение данных с помощью документов JSON или представлений «ключ-значение». Но их устаревшие данные по-прежнему хранятся в традиционной ORDBMS. Таким образом, изменение модели может повлиять на удобство использования запросов и приложений, разработанных в ORDBMS. Поэтому в унифицированной многомодельной базе данных исследовательская задача заключается в том, как выполнить отображение модели и переписывание запроса для автоматической обработки эволюции модели. Необходимо обратить внимание, что эволюция модели является более сложным процессом, чем эволюция схемы в ORDBMS, поскольку она включает в себя как изменение атрибутов, так и изменение структуры.

Можно отметить подход в статье [25]. Интеграция с глобальной схемой большинства существующих на сегодняшний день систем с несколькими базами данных обеспечивают обмен данными с помощью глобальной схемы. Эта схема обычно является результатом тесной и статичной интеграции нескольких схем.

При подходе к интеграции схем глобальная схема необходима для выполнения глобальных запросов. Обычно это означает переводы языков, схем или того и другого. Глобальный подход к схемам имеет два основных преимущества. С одной стороны, могут выполняться запросы, охватывающие множество баз данных, а с другой стороны, обеспечивается прозрачный доступ к целевым схемам. Обычно это достигается путем преобразования глобальной схемы в отдельные целевые схемы и путем предоставления средств для наложения псевдонимов.

До сих пор ни один автоматический перевод при обновлении и интеграции схемы не был успешным. Основным препятствием было понимание и интерпретация того, что означают различия, и каким-то образом перевод понимания из одной схемы в другую.

Еще одним препятствием является возможность поддерживать глобальные схемы в условиях частых изменений компонентной схемы. В существующих системах перевод и интеграция осуществляются на специальной основе. Поскольку базы данных в основном были внутренними, вопросы автономии не были важны. Таким образом, автономия баз данных не предусматривалась. Действительно, для того чтобы работать с такими тесно связанными базами данных, у отдельных баз данных не было другого выбора, кроме как раскрыть свою схему.

Поскольку существует единая централизованная схема, процесс принятия решений был централизован, и были устранены узкие места в производительности. На рисунке 3 показан подход к глобальной интеграции. Интеграция схем - это очень утомительная и требующая большого количества ручного труда работа. Она также подвержена ошибкам. Ошибка в проекте может привести к существенному нарушению целостности баз данных. Кроме того, любая модификация одной из схем участвующих баз данных может привести к серьезным изменениям в глобальной схеме, что, в свою очередь, может сделать участвующие базы данных недоступными на неприемлемый период времени.

Добавление или удаление схемы базы данных может привести к перепроектированию глобальной схемы с нуля, что может оказаться непомерно дорогостоящим. Сопоставление глобальной схемы с компонентными схемами также является проблематичным. Изменение глобальной схемы может привести к недопустимым изменениям в схемах компонентов. В настоящее время не существует автоматического способа распространения этих изменений сверху вниз или снизу вверх. Эта трудность обусловлена внутренней сложностью семантики данных. Хотя такой подход позволяет пользователям легко одновременно использовать несколько баз данных, для обеспечения однородности требуются большие накладные расходы.

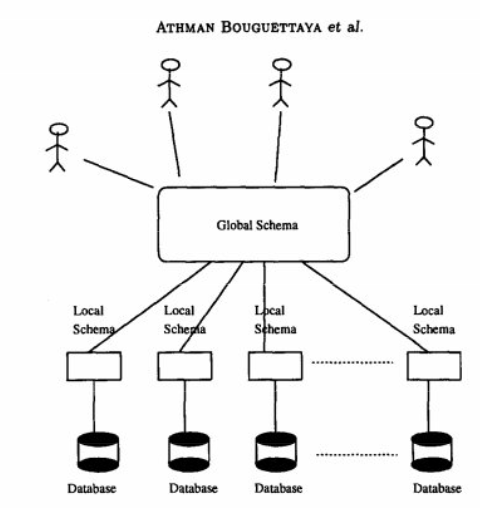


Рисунок 3. Схема взаимодействия

В небольшой сети разнородных баз данных и в рамках одной организации интеграция схем по-прежнему является жизнеспособным и разумным решением. Ожидается, что в этой среде пользователи будут знать, где находится информация в глобальной схеме. Это лучше всего подходит для тех сред, где все участвующие базы данных принадлежат одной организации и где обновления схем происходят нечасто.

При таком подходе представления являются хорошими инструментами для защиты данных и настройки их в соответствии с потребностями пользователей. С точки зрения пользователей, может оказаться неразумным предполагать, что пользователи знают всю схему целиком. Стоит отметить, что запросы к глобальной схеме не всегда могут быть детерминировано (и автоматически) преобразованы в подзапросы, которые охватывают несколько компонентных схем.

## Встроить на уровне системы

В статье [26] для решения данной проблемы предлагается 4-уровневой архитектуре клиент-сервер с использованием системы с несколькими базами данных можно визуализировать как систему клиент-сервер, которая позволяет клиентам одновременно получать доступ и обновлять данные, хранящиеся на нескольких серверах распределенных баз данных:

Уровень 1 — это клиентский графический пользовательский интерфейс или веб-интерфейс, который находится на вершине клиентской прикладной программы или сервера приложений

Уровень 2 — это сервер приложений, который содержит клиентскую программу, бизнес-логику, API и доступ к серверу системы с несколькими базами данных.

Уровень 3 — это система с несколькими базами данных, которая контролирует и поддерживает глобальную схему и глобальный каталог, а также доступ к различным удаленным серверам баз данных на основе запросов пользователей.

Уровень 4 — это удаленные гетерогенные локальные серверы компонентных баз данных.

## Графовое взаимодействие

В статье [27] авторы предлагают глобальную систему со следующими основными характеристиками:

* Графовый формализм (а именно GM) для представления и запроса баз данных. Этот формализм подходит для придания точной семантики сложным визуальным представлениям и является достаточно общим для формализации, в принципе, базы данных, выраженной в любой из наиболее распространенных моделей данных. Примитивы запроса формализма, хотя и состоят исключительно из двух элементарных графических действий, а именно выбора узла и рисования ребра, по крайней мере столь же выразительны, как реляционная алгебра.
* Адаптивный визуальный интерфейс, построенный на основе вышеуказанного формализма, предоставляющий пользователю различные визуальные представления и механизмы взаимодействия вместе с возможностью переключения между ними. Все различные визуальные представления позволяют выразить по крайней мере класс конъюнктивных запросов.
* Определение трех подходящих наборов алгоритмов перевода, один для перевода базы данных, выраженной в любой из наиболее распространенных моделей данных, во внутреннюю системную модель, один для перевода запроса GM в терминах языков запросов базовых моделей данных и один, предназначенный для реализации последовательного переключения между различными визуальными представлениями во время формулировки запроса.
* Построение и управление эффективной пользовательской моделью, которая позволяет системе предоставлять пользователю наиболее подходящее визуальное представление в соответствии с его навыками и потребностями.

Система состоит из диспетчера визуального интерфейса, диспетчера пользовательской модели, диспетчера GMDB и запросов и одной или нескольких СУБД. Диспетчер визуального интерфейса способен поддерживать несколько представлений (на основе форм, иконических, диаграммных и гибридных) баз данных и соответствующих модальностей взаимодействия. Таким образом, пользователю предоставляется язык многопарадигматических запросов, основанный на наборе визуальных языков запросов, каждый из которых взаимодействует с различным визуальным представлением GM и все они разделяют одну и ту же выразительную силу.

Представления на основе форм являются первой попыткой предоставить пользователю дружественные интерфейсы для манипулирования данными; они обычно предлагаются в рамках реляционной модели, где формы на самом деле являются таблицами. Их основная характеристика состоит в визуализации прототипических форм, в которых запросы формулируются путем заполнения соответствующих полей. В предшествующих системах, принявших представление на основе форм, таких как QBE, отображается только интенсиональная часть отношений: экстенсиональная часть заполняется пользователем, чтобы предоставить пример запрошенного результата.

В более поздних предложениях интенсиональная и экстенсиональная части базы данных сосуществуют. Диаграммные представления являются наиболее используемыми в существующих системах. Обычно они представляют с помощью различных визуальных элементов различные типы концепций, доступных в модели. Соответствие между визуальными элементами и связанными типами концепций требует эстетических критериев для размещения визуальных элементов и связей. Например, иерархические структуры для обобщения и агрегации объектов диктуют вертикальное размещение задействованных элементов. Диаграммные представления принимают в качестве типичных операторов запроса выбор элементов, обход смежных элементов и создание моста между разъединенными элементами.

Иконическое представление использует наборы иконок для обозначения как объектов базы данных, так и операций, которые должны быть выполнены с ними. Запрос выражается в первую очередь путем объединения иконок. Например, иконки могут быть объединены вертикально для обозначения конъюнкции (логическое И) и горизонтально для обозначения дизъюнкции (логическое ИЛИ). Чтобы быть эффективным, предлагаемый набор иконок должен быть легко понятен большинству людей.

Однако во многих случаях трудно или даже невозможно найти общепринятый набор иконок. В качестве альтернативы иконки могут быть определены пользователем для адаптации к конкретным потребностям пользователя и его/ее собственному ментальному представлению задач, которые он/она хочет выполнить. Гибридные представления используют произвольную комбинацию вышеуказанных подходов, либо предлагая пользователю различные альтернативные представления баз данных и запросов, либо объединяя различные визуальные структуры в единое представление. Диаграммы часто используются для описания схемы базы данных, в то время как значки используются либо для представления конкретных прототипических объектов, либо для указания действий, которые необходимо выполнить. Формы в основном используются для отображения результата запроса.

## Получение данных на основе JSON

В настоящее время многие основные реляционные базы данных, такие как Oracle, Microsoft SQL Server, MySQL, PostgreSQL и TeraData, активно изучаются для выявления способов оптимизации производительности базы данных для адаптации к эпохе больших данных. Таким образом, были предприняты попытки интегрировать хранилище текста JSON в реляционные базы данных для совместимости с базами данных NoSQL, тем самым достигая эффективного гибридного облачного хранилища.

Тем не менее, характеристики самих реляционных баз данных привели к их неотъемлемой неспособности выполнять обработку JSON. Этот недостаток также заставил разработчиков неохотно использовать единую реляционную базу данных для одновременной обработки высокопроизводительных данных и сложных логических реляционных данных в современной гибридной облачной системе хранения.

Более того, исследователи предложили несколько методов хранения текста JSON в реляционных базах данных. Обсуждалось хранение собственных данных JSON в коммерческих базах данных и использование SQL для расширенных запросов.

Авторами [28] предложен гибридный язык запросов JSON на основе SQL. Были предложены и сравнены два различных метода отображения, которые использовались для хранения данных JSON в реляционных базах данных.

Модель данных сущность-атрибут-значение использовалась для обсуждения поддержки двух реляционных баз данных с открытым исходным кодом и двух коммерческих реляционных баз данных для документов JSON.

В определенной степени интеграция текста JSON, хранящегося в реляционной базе данных, решила проблему взаимодействия различных типов данных в гибридном облачном хранилище. Однако реляционная база данных не подходит для текстового хранилища JSON, поскольку она в первую очередь предназначена для хранения реляционных структур данных. Более того, совместимость текста JSON, хранящегося в реляционных базах данных, зависит от его операторов SQL из-за ограничений его структуры. Поэтому многие исследователи пытались рассмотреть взаимодействие между различными типами данных в гибридном облачном хранилище с другой точки зрения, то есть достичь взаимодействия посредством взаимного сопоставления между JSON и реляционной базой данных для реализации единого управления хранилищем текста JSON и реляционной базой данных.

С точки зрения исследования сопоставления данных JSON с реляционными данными был предложен алгоритм сопоставления из JSON в реляционную базу данных, и данные JSON были сохранены в реляционной базе данных. JSON был определен в веб-запросе данных, и был проведен теоретический анализ для изучения метода ограничений целостности JSON. Была предложена формальная модель данных JSON и был определен легкий язык запросов. Был предложен формат обмена данными между службами RESTFul, который больше склонен хранить данные сетевых атрибутов.

На основе существующих моделей сопоставления между JSON и реляционными данными и нереляционными данными в статье были объединены эти две модели сопоставления и предложена новая модель JSON под названием XYJSON model. Используя сопоставление модели управления XYJSON, эта модель данных достигла унифицированного управления различными типами баз данных, помогая заполнить пробел в модели управления приложениями для гибридных облачных баз данных и продвигая исследования по унифицированному управлению для гибридных облачных баз данных.

# Объём рынка

Данные системы занимают довольно маленький объём рынка в сравнении с более распространёнными корпоративными решениями, такими как стандартные СУБД или облачные хранилища. В основном данные системы ориентированы на решение специфических задач.

Целевая аудитория данных систем — это, как правило, большие компании, работающие с множеством распределённых баз данных. Такие организации сталкиваются с необходимостью управления огромными массивами информации, поступающей из различных источников: транзакционных систем, CRM, ERP, IoT-устройств и аналитических платформ.

# Области применения

В биологии часто необходимо иметь в одном месте и анализировать данные из разных БД и СУБД, например, при изучении РНК, что весьма затрудняет работу специалистов [29, 30]. В решении данной проблемы помогает данная технология.

В политологии для анализа данных. Например, к настоящему времени в распоряжении исследователей имеется немало баз и наборов данных, которые содержат систематизированную информацию по административно-территориальным единицам (АТЕ) субнационального уровня, в том числе регионам. Такие данные собираются, систематизируются и публикуются официальными статистическими службами. Правда, практика их работы и формат представления информации существенно различаются в разных странах, но для стран-членов Евросоюза разнообразная информация по социально-эконмическим характеристикам регионов унифицирована в базе данных Евростата, статистического офиса ЕС. Для регионов европейских стран, не входящих в ЕС, подобную информацию можно найти в базе данных Организации экономического сотрудничества и развития [31].

Актуально в целом для любой большой компании для взаимодействия между разными частями, которые имею свои собственные СУБД и обособлены друг от друга.

# Заключение

В данной работе была подробно описана технология объединения нескольких систем управления базами данных (СУБД), Рассмотрены перспективы развития данной технологии, такие как повышение отказоустойчивости, масштабируемости и производительности распределённых систем.

Проведённый анализ демонстрирует, что объединение СУБД является перспективным направлением, способным решать сложные задачи обработки и хранения данных в условиях динамично развивающихся IT-инфраструктур.

# Список источников

1. Бердыбаев Р.Ш., Гнатюк С.О., Тынимбаев С., Азаров И.С. Анализ современных баз данных для использования в системах Siem // Вестник Алматинского университета энергетики и связи, Т. 54, №3, С. 33-47, 2021. DOI 10.51775/1999-9801\_2021\_54\_3\_33.
2. The Multi-model Databases – A Review / Ewa Płuciennik & Kamil Zgorzałek; — Springer, Cham: Communications in Computer and Information Science, vol. 716, 2017. — 141–152 с.
3. Key-Value stores:a practical overview / Marc Seeger; — Stuttgart, Germany, 2009.
4. Survey of graph database models / Renzo Angles, Claudio Gutierrez; — ACM Computing Surveys, vol. 40, 2008. — 1–39 с.
5. 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.
6. What is a Document Database? // mongodb.com – URL: https://www.mongodb.com/document-databases (дата обращения: 23.04.2025).
7. Multi-model database // wikipedia.org – URL: https://en.wikipedia.org/wiki/Multi-model\_database (дата обращения: 23.04.2025).
8. Мультимодельные СУБД — основа современных информационных систем? // habr.com – URL: https://habr.com/ru/articles/462493/ (дата обращения: 23.04.2025).
9. Data lake // en.wikipedia.org – URL: https://en.wikipedia.org/wiki/Data\_lake (дата обращения: 23.04.2025).
10. 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).
11. 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).
12. Pwint Phyu Khine, Zhao Shun Wang Data Lake: A New Ideology in Big Data Era. – Wuhan, China, 2017
13. Data Lake // yandex.cloud.ru – URL: https://yandex.cloud.ru/docs/glossary/datalake (дата обращения: 23.04.2025).
14. What is a Data Swamp? // medium.com – URL: https://medium.com/codex/what-is-a-data-swamp-38b1aed54dc6 (дата обращения: 23.04.2025).
15. 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.
16. Principles of Distributed Data Management in 2020? / Patrick Valduriez; — INRIA and LIRMM, Montpellier – France, 2020.
17. DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources / Laura Haas, Peter Schwarz, Prasad Kodali, Elon Kotlar, Julia Rice, William Swope; — IBM Systems Journal, vol. 40, 2001. — 489–511 c. — DOI: 10.1147/sj.402.0489.
18. MIKS: An Agent Framework Supporting Information Access and Integration / Domenico Beneventano, Sonia Bergamaschi, Gionata Gelati, Francesco Guerra, Maurizio Vincini; — Dipartimento Ingegneria dell’Informazione, Università di Modena, via Vignolese 905, 41100 Modena, Italy; CSITE-CNR, viale Risorgimento 2, 40136 Bologna, Italy.
19. hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems / Tian Luo, Rubao Lee, Michael Mesnier, Feng Chen, Xiaodong Zhang; — Proceedings of the VLDB Endowment, vol. 5, 2012.
20. The Study of Combining SQL and NoSQL Databases in a Heterogeneous System for the Development of a Project Management Database / O.V. Shalina, E.E. Andrianova; — Peter the Great St. Petersburg Polytechnic University; — Theoretical & Applied Science, 2024, no. 5 (133). — pp. 14-17.
21. Issues and Approaches of Database Integration / Christine Parent, Stefano Spaccapietra; — Communications of the ACM, vol. 41, no. 5es, May 1998. — pp. 166–178. — DOI: 10.1145/276404.276408.
22. Design of an Integrated DBMS to Support Advanced Applications / V. Lum et al., In: A. Blaser, P. Pistor, Datenbank-Systeme für Büro, Technik und Wissenschaft. Informatik-Fachberichte, vol. 94; — Springer, Berlin, Heidelberg, 1985. — DOI: 10.1007/978-3-642-70284-6\_26.
23. Automation and Orchestration Framework for Large-Scale Enterprise Cloud Migration / J. Hwang, K. Bai, M. Tacci, M. Vukovic and N. Anerousis; — IBM Journal of Research and Development, vol. 60, no. 2-3, 2016. — 1:1–1:12 c.
24. UDBMS: Road to Unification for Multi-model Data Management / Jiaheng Lu, Zhen Liu, Pengfei Xu, Chao Zhang; — In: Advances in Databases and Information Systems (ADBIS), 2018. — 330–344 c. — DOI: 10.1007/978-3-030-01391-2\_33.
25. On Building a Hyperdistributed Database / Michael Papazoglou, Athman Bouguettaya; — Information Systems, vol. 20, no. 5, 1995. — DOI: 10.1016/0306-4379(95)00030-8.
26. A Multidatabase System as 4-Tiered Client-Server Distributed Heterogeneous Database System / Ali Ghulam; — International Journal of Computer Science and Information Security, vol. 6, 2009. — 010–014 c. — DOI: 10.48550/arXiv.0912.0579.
27. Graphical Interaction with Heterogeneous Databases / T. Catarci, G. Santucci, J. Cardiff; — The VLDB Journal, vol. 6, 1997. — 97–120 c. — DOI: 10.1007/s007780050035.
28. JSON-Based Control Model for SQL and NoSQL Data Conversion in Hybrid Cloud Database / Lei Zhang, Ke Pang, Jiangtao Xu, Bingxin Niu; — Journal of Cloud Computing, vol. 11, 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.