СОДЕРЖАНИЕ ЛЕКЦИИ 1

Введение. Обзор технологий баз данных. Эволюция технологий БД. Краткая история моделирования данных. Первые БД. Реляционные базы данных. Особенности работы реляционных БД.

Введение. Обзор технологий баз данных

Введение в базы данных

База данных (БД) представляет собой организованный набор информации, спроектированный для эффективного хранения, доступа, обновления и управления данными компьютерной системой.

Базы данных находят широкое применение в различных сферах, включая бизнес, образование, здравоохранение, науку, государственное управление и другие области. Они играют важную роль в современном мире, обеспечивая эффективное хранение, организацию и обработку данных, что необходимо для успешной деятельности в различных областях.

До появления современных баз данных, информация для программных систем чаще всего хранилась в плоских[1] текстовых файлах, бинарных файлах, файлах с разделителями типа CSV (CHARACTER-SEPARATED VALUES) или в файлах с уникальными форматами, разработанными для конкретных приложений. Это создавало ограничения для эффективного доступа к данным, их поиска и обработки, затрудняло масштабирование и поддержку программ.

Появление баз данных открыло новые возможности для хранения данных в универсальных структурированных форматах, что значительно улучшило эффективность и гибкость обработки информации. Базы данных позволяют проводить различные операции над данными, такие как поиск,

[1] Понятие «плоские» отражает то, что данные хранятся в виде линейного потока информации, часто в виде строк текста, без иерархии или связей между ними

фильтрация, сортировка, агрегация, анализ и обновление, что позволяет извлекать ценные знания из данных и принимать обоснованные решения.

Одна из ключевых особенностей баз данных – возможность одновременной работы с данными множеством пользователей, обеспечивая контроль доступа и координацию изменений. Они также предоставляют средства для защиты данных от несанкционированного доступа, включая механизмы аутентификации, авторизации, шифрования и разграничения доступа.

Современные базы данных обладают высокой масштабируемостью и гибкостью, что позволяет им обрабатывать как небольшие, так и огромные объемы данных, а также приспосабливаться к различным требованиям и сценариям использования.

Роль баз данных в различных сферах деятельности невозможно переоценить. Они помогают бизнесу эффективно управлять клиентской информацией, автоматизировать операции и принимать обоснованные решения. В научных исследованиях они обеспечивают обработку больших объемов данных и совместную работу ученых. В учебных заведениях базы данных помогают в управлении учебными процессами и создании образовательных ресурсов. А правительственным организациям они предоставляют данные для анализа и разработки политик и программ.

Эволюция технологий БД

Под технологиями баз данных понимается совокупность программных и аппаратных средств, а также методов и подходов к организации данных, которые обеспечивают структурированное и эффективное хранение, доступ и управление данными в информационной системе. Эти технологии включают в себя следующие компоненты:

Системы управления базами данных (СУБД) – специализированное программное обеспечение, разработанное для создания, управления и манипулирования базами данных. Английская аббревиатура – DBMS (Database Management System). СУБД предоставляют программам удобные способы взаимодействия (интерфейсы) для выполнения операций с данными, скрывая сложности взаимодействия с физическим хранилищем данных. Программы могут обращаться к данным через СУБД, используя специализированные запросы и команды, а СУБД занимается обработкой этих запросов и обеспечением безопасности и целостности данных в базе данных. Примеры современных СУБД: Oracle, Microsoft SQL Server, PostgreSQL, MySQL.

Следует отметить, что в разговорной речи и литературе часто используются оба термина «базы данных» и «системы управления базами данных», чтобы обозначить одну и ту же концепцию – программное обеспечение, предназначенное для хранения, управления и обработки данных. В ряде случаев можно считать эти термины взаимозаменяемыми.

Языки запросов – специализированные языки, используемые для взаимодействия с данными в базах данных. Они позволяют пользователю выполнять различные операции, такие как извлечение данных, добавление новых записей, обновление существующих записей и удаление данных из базы данных. Кроме формулирования запросов к базе данных эти языки поддерживают команды по созданию баз данных, определению и редактированию их структуры, управлению доступом к данным. Они обладают строгим синтаксисом и специальными конструкциями для работы с данными, обеспечивая удобный и эффективный способ взаимодействия с базой данных. Наиболее распространенным языком запросов является язык SQL (Structured Query Language).

Методы организации данных – техники, определяющие способы структурирования и хранения данных в базе данных для обеспечения эффективного доступа к ним. Эти методы охватывают разнообразные модели

данных, включая реляционную, иерархическую, сетевую, объектно-ориентированные и другие.

Эволюция технологий баз данных (БД) прошла через несколько значительных этапов, от ранних систем управления данными до современных распределенных и облачных решений.

В начале развития технологий хранения и обработки данных последние организовывались в виде иерархических и сетевых структур. Эти модели были довольно ограниченными в своей гибкости и требовали сложных запросов для доступа к информации.

В 1970-х Эдгар Кодд предложил реляционную модель данных, основанную на теории множеств и логике предикатов, и которая впоследствии стала основой современных систем управления базами данных (СУБД). Реляционные базы данных используют таблицы, связанные ключами, что делает их более гибкими и понятными для разработчиков и пользователей. Стандартом для работы с реляционными базами данных стал язык структурированных запросов (SQL – Structured Query Language). Он предоставляет универсальный набор инструкций для создания, изменения, и извлечения данных.

В ответ на растущие потребности в хранении и обработке сложных структур данных и благодаря развитию технологий программирования в 80-90-х годах стали разрабатываться объектно-ориентированные базы данных. Они позволяют хранить объекты, такие как классы и экземпляры, непосредственно в базе данных.

С начала 2000-х стали приобретать популярность и другие нереляционные базы данных, основанные на альтернативных моделях, таких как документы, столбцы и графы. Их появление обусловлено ростом объемов данных, высоким требованиям к скорости обработки и распространением полуструктурированных (например, JSON, XML и др) и неструктурированных (текстовые, аудио, видео) данных.

Облачные базы данных предоставляют возможность хранить и обрабатывать данные в облачной инфраструктуре, такой как Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform и другие. Это удобно для компаний, которым необходима масштабируемость и гибкость ресурсов без необходимости инвестировать в собственную инфраструктуру.

С появлением больших объемов данных, таких как «большие данные» (Big Data), «интернет вещей» (IoT – Internet of Things) и других источников данных, стандартные базы данных столкнулись с проблемами масштабируемости и производительности. Это послужило толчком к появлению распределенных базах данных, которые хранят данные на нескольких компьютерах (или узлах) высокоскоростной сети как в облаке, так и в локальной инфраструктуре. Распределенные базы данных обеспечивают высокую доступность и масштабируемость данных, что позволяет эффективно обрабатывать большие объемы информации.

Блокчейн, хотя и не всегда рассматривается как типичная база данных, представляет собой децентрализованную и надежную систему записи, хранения и обработки транзакций[2]. Он нашел применение в финансовой сфере, управлении поставками, телекоммуникациях и других областях, где требуется надежная и эффективная система управления данными.

Все перечисленные этапы демонстрируют, как технологии баз данных за восемь десятилетий эволюционировали от простых иерархических моделей до сложных, распределенных систем, способных обрабатывать огромные объемы разнообразных данных.

Это лишь краткая история развития технологий баз данных, которые продолжают развиваться и изменяться в соответствии с технологическими и индустриальными требованиями.

[2] Транзакция – это совокупность запросов к БД, реализуемых

Понятие модели данных. Краткая история моделирования данных

Модель данных – это абстрактное представление объектов предметной области для организации и хранения информации в базе данных, а также для эффективного доступа к данным. Модели данных служат основой для проектирования баз данных, информационных систем и приложений. Единые правила для организации данных облегчают их использование как внутри программной системы между её компонентами, так и между различными программными системами, способствуя согласованности и стандартизации представления данных.

Развитие моделей данных шло параллельно с развитием языков программирования высокого уровня и структур данных, которые поддерживались этими языками. Появление абстрактного и удобного представления данных, таких как массивы, структуры, записи, объединения и объекты, позволяло программистам лучше организовывать данные в своих программах и более эффективно их обрабатывать. В ответ на растущие потребности в долгосрочном хранении информации появились универсальные способы организации и хранения данных – файлы. Форматы структурированных файлов являются своего рода предшественниками моделей данных, их можно рассматривать как простейшие формы организации для хранения данных.

Рассмотрим пример организации линейного текстового файла, хранящего информацию о книгах в библиотеке. В этом примере каждая книга представлена в текстовом формате без явной структуры или разделителей между полями, однако файл обладает некоторой структурой, поскольку порядок информации о книгах однозначно определен: сначала идет название, затем автор, год издания и ISBN и такое описание повторяется для каждой книги. Таким образом, данный файл можно рассматривать как частично структурированный, хотя он не полностью соответствует структурированному формату данных.

```
Война и мир
Лев Толстой
Роман
1869
ISBN978-5-17-119072-3

Преступление и наказание
Федор Достоевский
Роман
1866
ISBN978-5-17-100732-6

Горе от ума
Александр Грибоедов
Комедия
1825
ISBN978-5-17-000245-4
```

Следующий пример – структурированный файл в одном из популярных форматов CSV. В этом примере каждая строка представляет собой отдельную книгу, а различные атрибуты книги (название, автор, жанр, год издания, ISBN) разделены запятыми. Такой формат данных удобен для хранения табличных данных, и его легко создавать и интерпретировать компьютерными программами:

```
Название,Автор,Жанр,Год издания,ISBN
"Война и мир","Лев Толстой","Роман",1869,ISBN978-5-17-119072-3
"Преступление и наказание","Федор Достоевский","Роман",ISBN1866,978-5-17-1007
"Горе от ума","Александр Грибоедов","Комедия",1825,ISBN978-5-17-000245-4
```

Еще один формат данных, который можно назвать форматом таблицы с фиксированной шириной (Fixed Width Table Format):

```
Название книги    |Автор                |Жанр      |Год    |ISBN
------------------+---------------------+----------+-------+---------------
Война и мир       | Лев Толстой         | Роман    | 1869  |978-5-17-1072-3
Идиот             | Федор Достоевский   | Роман    | 1866  |978-5-17-1007-6
```

В различных отраслях использовались специфичные форматы данных, разработанные для конкретных приложений или систем. Например, в банковском секторе до сих пор используются форматы обмена данными, такие как SWIFT (Society for Worldwide Interbank Financial Telecommunication), FIX (Financial Information eXchange) и др., хотя они имеют ограничения гибкости и сложности в обработке и интерпретации данных, по сравнению с более современными форматами, такими как XML и JSON.

Подобные способы организации просты в создании и понимании, однако они слабо пригодны для автоматизированной обработки: неудобны для представления сложных или вложенных структур данных, не предоставляют встроенных механизмов для обеспечения безопасности и целостности данных, не пригодны для хранения больших объемов данных, эффективного доступа и анализа.

В ответ на растущие потребности в обработке все бóльших объемов информации, требовались единые стандартизованные унифицированные способы организации данных, методы доступа к этим данным, объединённеы в понятии модели данных.

Модель данных обычно включает в себя следующие аспекты:

Структура данных – это способ организации данных в базе данных в форме таблиц, деревьев, графов или других абстрактных структур, а также типы данных, которые определяют формат и ограничения для каждого элемента в этой структуре.

Связи между данными – это описание того, как данные взаимосвязаны между собой, каковы типы и характеристики этих взаимосвязей, а также правила их создания.

Ограничения целостности – набор правил или условий, которые должны соблюдаться при вставке, обновлении и удалении данных, чтобы

гарантировать сохранение структуры и связей, избежать дублирования и предотвратить появление некорректных или противоречивых данных.

Язык запросов – это набор инструкций или команд, которые позволяют пользователю взаимодействовать с базой данных: извлекать нужную информацию, модифицировать данные или выполнять другие операции. Эти команды могут быть заложены в программную систему или формулироваться пользователем при работе с программой, часто неявным, интуитивно понятным способом.

Текстовые файлы, как неструктурированные, так и структурированные, можно рассматривать как простейшие формы организации для хранения данных. Они являются своего рода предшественниками моделей данных, поскольку предоставляют базовый способ структурирования. Однако они не определяют единый способ управления данными, основными функциям которого являются CRUD-операции:

Create (создание) - создание новых записей или объектов в базе данных;

Read (чтение) - чтение или извлечение информации из базы данных;

Update (обновление) - обновление существующих записей или объектов в базе данных;

Delete (удаление) - удаление существующих записей или объектов из базы данных.

Поэтому понятие модели данных обычно связывается с более сложными и унифицированными способами организации и представления данных, а также едиными методами управления этими данными. Наиболее известными моделями данных являются:

– Иерархическая модель – одна из первых моделей, в которой данные организованы в иерархической структуре, состоящей из уровней, каждый из которых может содержать связанные между собой записи.

– Сетевая модель, появившаяся после иерархической модели, и позволяющая более гибко организовывать данные, разрешая им иметь «неиерархические» связи в дополнение к иерархическим.

– Реляционная модель, внедрение которой стало переломным моментом в развитии баз данных за счет базирования на теории множеств и логике предикатов, использования таблиц и связей между ними для представления данных.

– Объектно-ориентированная модель, появившаяся практически одновременно с развитием объектно-ориентированного программирования и позволяющая хранить объекты непосредственно в базе данных.

– Специализированные NoSQL модели, появившиеся в последние десятилетия, предлагающие альтернативные подходы к организации данных, особенно в случае неструктурированных данных и больших объемов информации.

Каждая модель данных имеет свои преимущества и недостатки, и выбор модели зависит от конкретных потребностей и задач приложения. Правильно спроектированная модель может улучшить производительность системы, оптимизировать запросы к данным и обеспечить эффективное использование ресурсов хранения и обработки.

Моделирование данных – это процесс создания абстрактного представления данных и их взаимосвязей в рамках конкретной предметной области при проектировании программной системы. Этот процесс может быть как ручным, так и автоматизированным. История моделирования данных охватывает эволюцию методов и инструментов, используемых для описания и проектирования структуры данных в базах данных.

В прошлом модели данных обычно создавались и документировались вручную с помощью бумаги, карандашей и линеек. Этот процесс мог быть довольно трудоемким и подверженным ошибкам, особенно при работе с большими и сложными программными системами.

В настоящее время модели данных создаются с использованием специализированных программных средств, таких как программы для проектирования реляционных баз данных (например, ER-диаграммы), CASE-средства (Computer-Aided Software Engineering) и другие графические и текстовые редакторы. Эти инструменты значительно упрощают процесс моделирования данных, обеспечивая более высокую точность и эффективность. Современные инструменты моделирования данных помогают разработчикам и аналитикам лучше понять и описать требования к данным и дизайн крупных и сложных баз данных до их реальной реализации в программных системах. Широкие возможности визуализации и документирования моделей, поддержка совместной работы в рамках команды способствуют лучшему взаимопониманию между разработчиками, аналитиками и другими участниками крупных программных проектов и упрощают процесс обмена информацией о проектируемой базе данных.

На нынешнем этапе развития инструменты моделирования данных не только обеспечивают возможности для создания и анализа моделей, но также способны автоматически генерировать на их основе программный код. Это позволяет быстро переводить проектирование баз данных в рабочее программное обеспечение, что значительно экономит время и силы разработчиков. Автоматическая генерация кода на основе моделей данных обеспечивает согласованность между архитектурой базы данных и приложением, а также уменьшает вероятность ошибок при реализации.

С ростом популярности NoSQL технологий возникла потребность в соответствующих инструментах для создания баз данных нетрадиционной структуры. Как правило, компании-разработчики NoSQL-СУБД предоставляют свои собственные инструменты, ориентированные на особенности своих моделей данных. Эти инструменты поддерживают как текстовый, так и графический интерфейс, предоставляют богатый набор функций для проектирования, анализа, мониторинга и оптимизации баз данных.

Таким образом, современные инструменты моделирования данных играют важную роль в разработке информационных систем и обеспечивают более эффективный и надежный процесс проектирования баз данных.

Первые иерархические и сетевые базы данных

Иерархические и сетевые модели данных первыми легли в основу систем управления данными, появившимися во второй половине XX века. А иерархические и сетевые БД были разработаны для хранения и организации информации в соответствии с этими моделями.

В иерархической базе данных данные организованы в виде древовидной структуры (перевернутое дерево), где каждый элемент, кроме корневого, имеет только одного родителя, но может иметь несколько дочерних элементов. Иерархическая модель представляет собой ориентированный граф (рисунок 1).

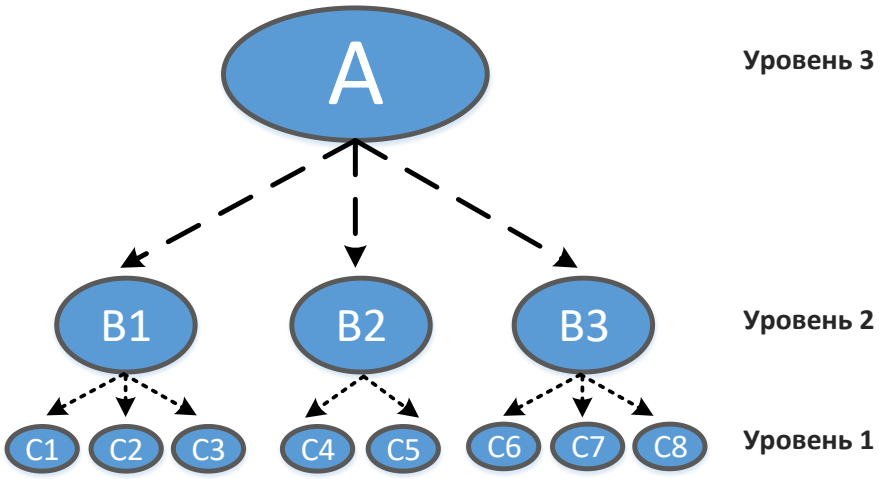

Рисунок 1 – Иерархическая модель данных

Такая модель хорошо подходит для организации данных с явно выраженной иерархией, например, организационная структура компании, учебная образовательная система, файловая система компьютера. Пример

иерархической модели финансовой организации – банка – представлен на рисунке 2:

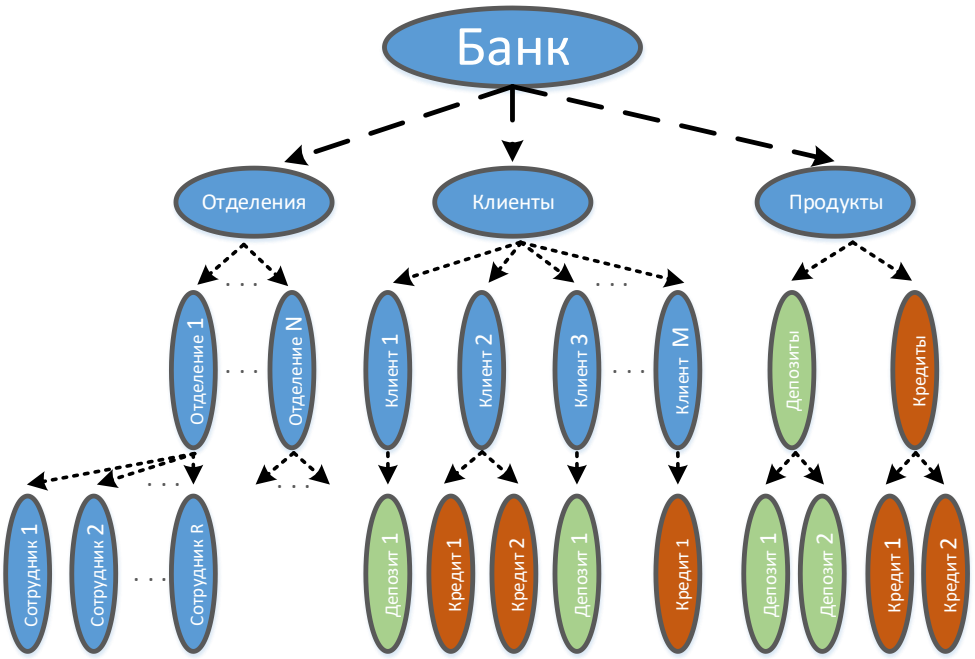

Рисунок 2 – Фрагмент иерархической модели банка

В этом примере «Банк» является корневым элементом древовидной структуры. У него есть дочерние элементы, такие как «Отделения», «Клиенты» и «Продукты». В каждом отделении свой штат сотрудников. Каждый клиент банка может иметь несколько счетов и займов. Каждый банковский продукт (например, депозиты или кредиты) также может иметь свои дочерние элементы. Недостатки модели: невозможно избежать дублирования информации о продуктах. Нельзя одному работнику работать в двух отделах (совместительство). Если потребуется добавить новый тип информации или изменить связи между элементами, это может потребовать значительных изменений в структуре всей базы данных.

Одной из самых известных систем управления иерархическими базами данных, которая использовалась для управления данными в финансовых системах, авиаперевозках и других критически важных областях, является система IMS (Information Management System), разработанная компанией

IBM в 60-х годах прошлого столетия. IMS имеет ключевое значение в развитии современных баз данных и до сих пор широко используется в крупных предприятиях, таких как финансовые институты, авиакомпании, страховые компании и другие. Это связано с высокой производительностью, надежностью и масштабируемостью.

Запросы в IMS реализуются с использованием языка запросов DL/I (Data Language/I), специфичного для IMS. Ниже приведены некоторые типичные запросы, на этом языке:

```
#Запрос на получение данных из базы данных – операция чтения:
GET EMPLOYEE WITH EMPLOYEE-ID = '123456'

#Запрос на добавление новых данных
STORE NEW-RECORD IN CUSTOMER

#Запрос на обновление существующих данных  - операция замены
REPLACE PHONE-NUMBER OF CUSTOMER WITH CUST-ID = '123456' TO '555-5555'

#Запрос на удаление данных – операция удаления
DELETE EMPLOYEE WITH EMPLOYEE-ID = '123456'
```

IMS продолжает использоваться во многих крупных предприятиях и организациях по всему миру. Эта СУБД постоянно развивается, внедряя новые технологии и функции для соответствия современным требованиям бизнеса, такие как технологии виртуализации, облачных вычислений и аналитики данных. Многие предприятия уже многие годы успешно используют IMS и интегрируют его с другими системами и приложениями.

Хотя точный список предприятий, использующих IMS, не всегда общедоступен из-за конфиденциальности информации о применяемых системах и технологиях, в него включаются крупные банки и финансовые институты, такие как JP Morgan Chase, Bank of America и Wells Fargo, а также крупные транспортные компании, страховые компании и некоторые государственные организации и агентства.

В сетевой модели данные организованы в виде графа, где каждый элемент может иметь несколько связей с другими элементами, что позволяет более гибко описывать сложные отношения (рисунок 3). Это отличие от предыдущей модели данных, где данные организовывались в виде иерархии с родительскими и дочерними элементами, и стало одним из основных направлений в развитии моделей данных в 1960-1970 годах.

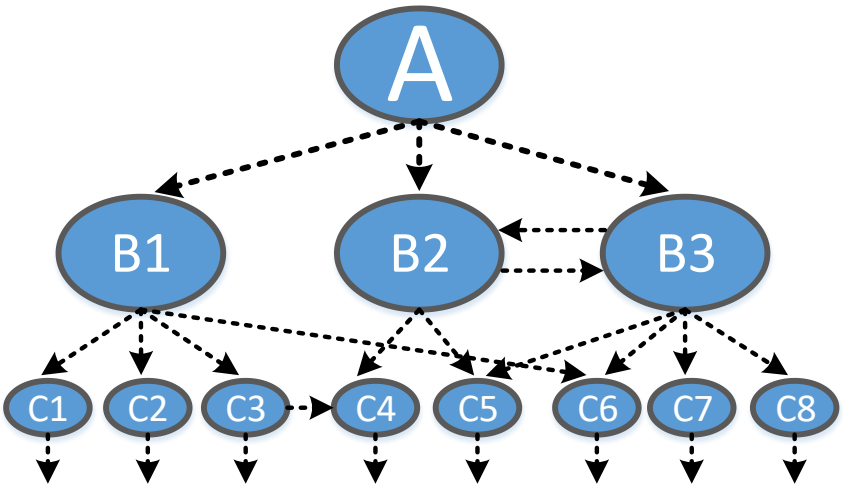

Рисунок 3 – Сетевая модель данных

Для реализации и поддержки сетевой модели в 1969 году был разработан стандарт CODASYL (Conference on Data Systems Languages). Его можно рассматривать как набор правил и спецификаций, описывающих структуру и доступ к данным в базе данных. В сетевой структуре CODASYL данные организованы в виде сети записей, связанных между собой. Основной элемент структуры данных – запись (record), которая может иметь несколько связей (set) с другими записями. Каждая связь имеет имя и указатель на набор записей. Такой подход к организации данных позволяет моделировать сложные отношения и обеспечивает более гибкую структуру, по сравнению со структурой иерархических баз данных.

Сетевая модель данных нашла применение в научных и инженерных областях, где требовалась поддержка сложных структур данных, таких как графы связей в научных исследованиях, инженерные системы и процессы,

например, сети передачи данных, электрические цепи, телекоммуникационные сети и другие. В банковской сфере сетевая модель позволяла описывать сложные структуры данных, включая связи между банковскими счетами, операциями, кредитами и клиентскими данными. Пример модели, которая позволяет учитывать разные виды отношений между клиентами банка, их счетами и кредитами и обладает, на рисунке 4.

Конкретные примеры систем управления данными, построенных на стандарте CODASYL, менее известны, чем системы, основанные на более распространенных моделях, таких как иерархическая или реляционная, но они все же существовали и использовались в определенных областях и индустриях. К ним можно отнести коммерчески доступные СУБД IDS (Integrated Data Store) и DMS-1100 (Data Management System), а также язык запросов и манипулирования данными NDML (Network Data Management Language).

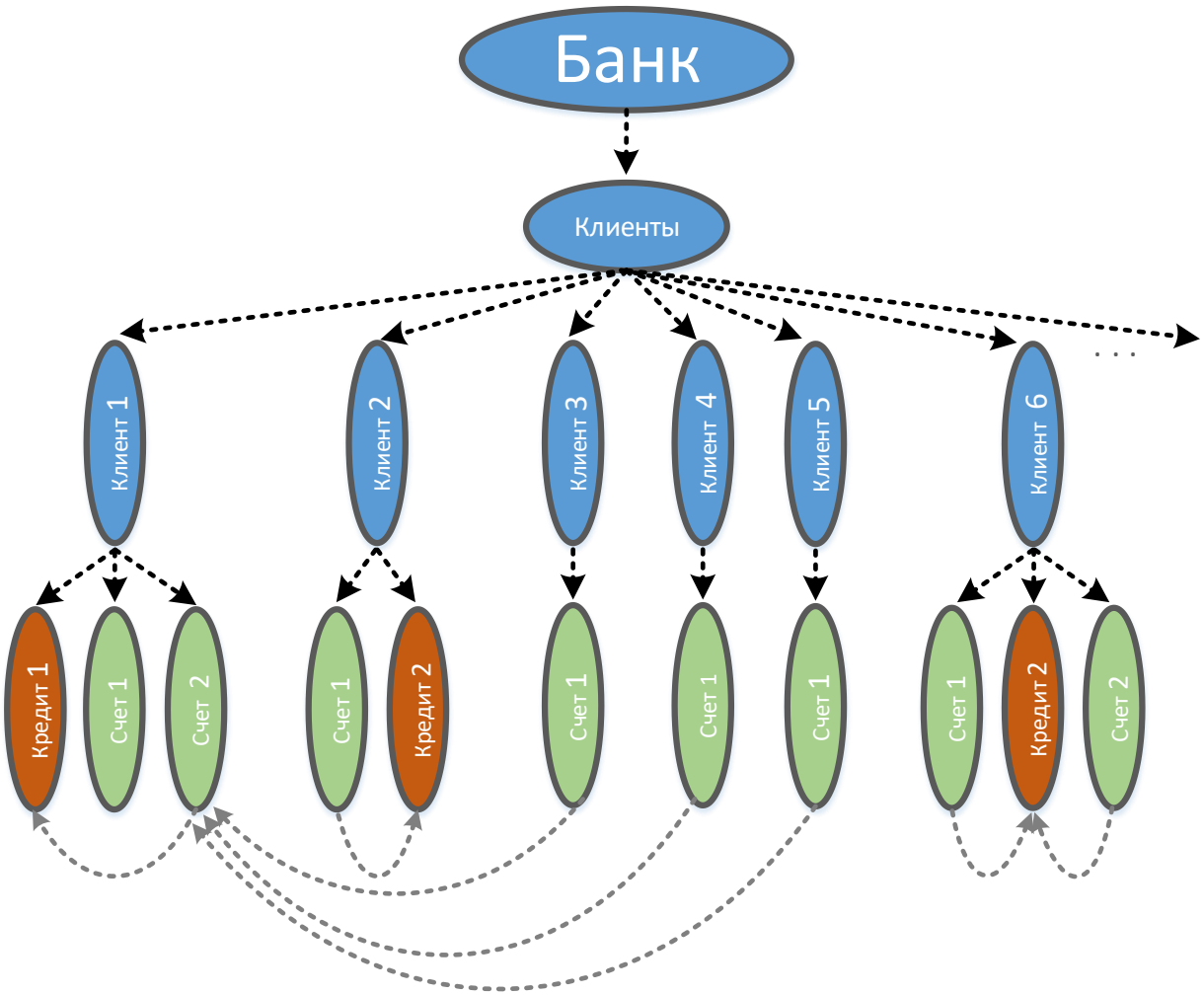

Рисунок 4 – Фрагмент сетевой модели банка

Основная цель CODASYL заключалась в создании универсального языка для определения и манипулирования данными в базах данных, а также в создании более гибких структур данных для организации хранения информации. Это было шагом вперед по сравнению с ограничениями иерархической модели. Однако использование структур данных, таких как множества, записи, связи и ключи, делало сетевую модель достаточно сложной для проектирования и обслуживания. Специальный язык запросов не всегда был удобным или интуитивно понятным для пользователей, что создавало сложности при выполнении запросов к данным. Базы данных CODASYL был менее гибким в отношении масштабирования и изменения

структуры данных. И хотя CODASYL внес существенный вклад в развитие сетевых моделей данных, с течением времени его популярность снизилась. Ограничения по гибкости и сложности запросов стали мотивацией для разработки более эффективных и удобных моделей данных, таких как реляционные базы данных.

Завершая обзор иерархических и сетевых БД и их моделей, следует подчеркнуть, что IMS и СУБД на основе CODASYL были мощными и эффективными системами управления данными своего времени. Они, вместе с другими иерархическими и сетевыми системами, играли важную роль в развитии хранения и организации данных, выступая в качестве прародителей или предшественников реляционных баз данных, и помогли сформировать понимание о том, как хранить и организовывать данные в информационных системах. Несмотря на утрату популярности, иерархические и сетевые модели данных вместе со своими системами управления данными продолжают оставаться значимыми вехами в истории развития баз данных

Реляционные базы данных

Реляционная модель

Иерархические и сетевые модели данных, несмотря на свои ограничения, вдохновили многих специалистов в области управления данными на дальнейшие исследования и разработки. Разработчики и инженеры направили усилия на создание более эффективных и удобных методов управления информацией. В результате появилась реляционная модель данных, предложенная Эдгаром Коддом в 1970 году, ставшая основой для многих современных систем управления базами данных. Термин «реляционный» означает, что теория баз данных основана на абстрактном

математическом понятии «отношения» (англ. relation). В качестве неформального синонима отношения часто используется слово таблица.

Реляционная модель данных представляет собой структуру организации данных в виде таблиц (отношений), которые состоят из строк (записей или кортежей) и столбцов (полей или атрибутов). Каждая таблица представляет собой отдельную сущность, каждая строка таблицы содержит конкретный экземпляр этой сущности, а каждый столбец представляет собой атрибут или характеристику этой сущности.

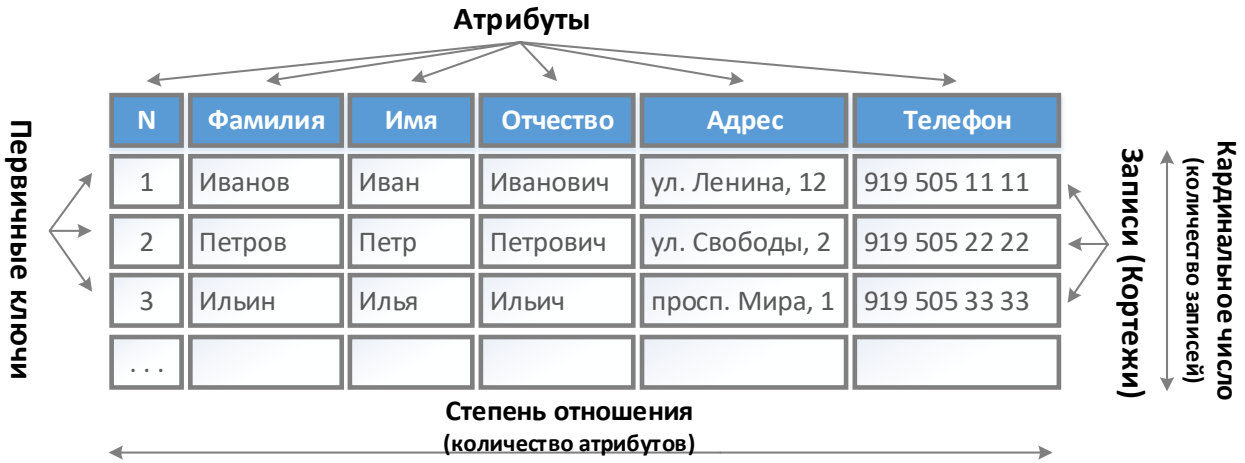

Рисунок 4 – Реляционная таблица

Экземпляры сущности имеют особый атрибут – так называемый первичный ключ PK (PPRIMARY KEY), который однозначно идентифицирует эту запись в БД. Помимо первичных, у записей могут быть и внешние ключи FK (FOREIGN KEY), с помощью которых реализуются связи между сущностями из разных таблиц. В реляционной модели есть понятие родительской и дочерней таблиц. Дочерняя таблица содержит один и более внешних ключей, которые есть не что иное, как первичные ключи родительских таблиц. Связи в реляционной модели обладают большей гибкостью по сравнению со связями в иерархической и сетевой моделях.

В реляционной модели существует три типа связи, отражающих отношение множественности.

«Один к одному» (one-to-one). В этом типе связи каждый экземпляр одной сущности связан с единственным экземпляром другой сущности. Например, каждый сотрудник имеет только одну карточку доступа к офису, а каждая карточка доступа принадлежит только одному сотруднику.

«Один ко многим» (one-to-many): Здесь каждый экземпляр одной сущности может быть связан с несколькими экземплярами другой сущности. Например, каждый сотрудник может иметь много больничных листов, но больничный лист принадлежит только одному сотруднику.

«Многие ко многим» (many-to-many): В этом случае каждый экземпляр одной сущности может быть связан с несколькими экземплярами другой сущности, и наоборот. Например, каждый сотрудник может участвовать в нескольких проектах, и над каждым проектом может работать несколько сотрудников.

Однако в плоской (или двумерной) реляционной модели данных каждый элемент таблицы должен иметь фиксированный набор полей, что делает невозможным представление множественных связей между элементами. Необходимо разрешить (решить) эту проблему. Термин «разрешение связи многие-ко-многим» означает преобразование такой сложной связи в более простые связи, которые могут быть реализованы в рамках плоской модели данных. Это разрешение реализуется через дополнительную таблицу, которая связывает экземпляры из обеих сущностей. Вместо одной связи «многие-ко-многим» появляются две связи «один-ко-многим», каждая из которых связывает исходную сущность с вспомогательной таблицей. В свою очередь, вспомогательная таблица содержит два столбца, которые представляют собой внешние ключи, связанные с каждой из исходных сущностей. Такое преобразование позволяет управлять связями более гибко и эффективно. Оно также позволяет проводить операции вставки, обновления и удаления данных более эффективно, так как операции выполняются на уровне «один-ко-многим», что упрощает манипуляции с данными.

В контексте отношения «один ко многим» родительская таблица будет иметь связь «один», а дочерняя таблица – «много». В отношении «один к одному» роль родительской и дочерней таблицы не так очевидна, однако по-прежнему дочерняя та, которая ссылается на родительскую через внешний ключ.

Примеры связей между сущностями в реляционной модели приведены на рисунке 5.

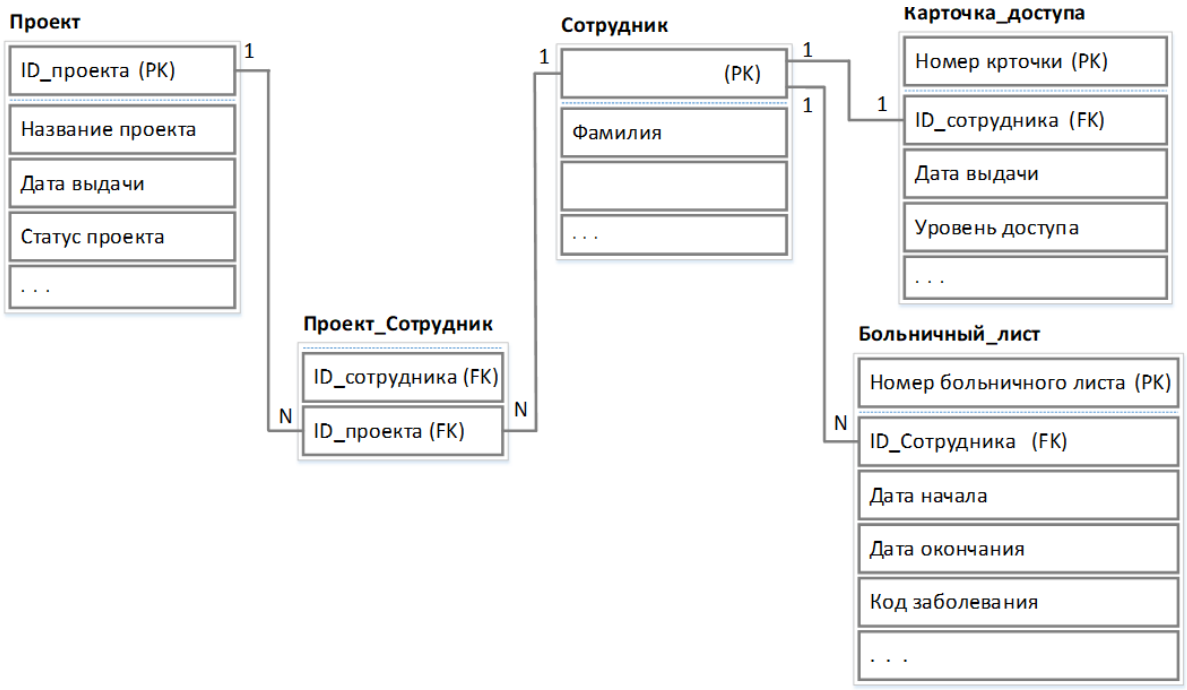

Рисунок 5 – Примеры связей между сущностями

Кроме множественности, связи между таблицами характеризуются обязательностью. Обязательность связи определяет, может ли значение внешнего ключа быть NULL или обязательно должно ссылаться на существующую запись в родительской таблице. Например, если в связи между таблицами Сотрудник и Карточка_доступа установлена обязательность связи, это означает, что каждый сотрудник обязан иметь карту доступа, и не существует карты доступа без сотрудника.

Еще один важный аспект связи в реляционной модели – тип каскадного действия (CASCADING ACTION) или правила обновления и удаления. Эти правила определяют, что происходит со связанными записями в связанных

таблицах при обновлении или удалении записей в основной (родительской) таблице. Основные типы каскадных действий представлены в таблице 1.

Таблица 1 – Типы каскадного взаимодействия связанных таблиц

Тип каскадного действия	Действия при обновлении или удалении записи в основной таблице
CASCADE	связанные записи в дочерних таблицах также обновляются или удаляются автоматически
SET NULL	внешний ключ в связанной таблице устанавливается в NULL
SET DEFAULT	внешний ключ в связанной таблице устанавливается в значение по умолчанию
RESTRICT	ограничивает операции обновления и удаления в основной таблице, если существуют связанные записи в дочерних таблицах.
NO ACTION	Это похоже на RESTRICT, но при этом операции обновления или удаления в основной таблице просто отменяются

Реляционная модель данных предоставляет декларативный (описательный) язык запросов SQL, который значительно упрощает выполнение операций над данными. А всю работу по выполнению запросов, поиску данных в таблицах, а также обеспечению целостности данных с помощью ограничений и транзакций выполняет СУБД. Она обрабатывает SQL-запросы, оптимизирует их выполнение, управляет доступом к данным, а также обеспечивает надежность и целостность хранимой информации.

SQL-запрос – это специальная команда или инструкция, которая отправляется базе данных со стороны приложения, чтобы выполнить определенное действие или получить определенные данные из базы данных. SQL-запросы могут выполнять различные операции, такие как добавление, удаление, изменение и выборка данных, в зависимости от поставленной цели.

Например, если необходимо получить информацию о пользователях из таблицы «Пользователи», SQL-запрос может выглядеть примерно так:

```
SELECT * FROM Users;
```

Этот запрос выберет все строки и столбцы (для этого используется символ *) из таблицы «Пользователи» и вернет их в программу, посредством которой пользователь обращается к базе данных. Каждый SQL-запрос обычно начинается с ключевого слова, которое указывает на тип операции (например, SELECT, INSERT, UPDATE, DELETE), за которым следует дополнительная информация, такая как имена таблиц, условия фильтрации и т. д.

Итак, SQL-запросы - это способ взаимодействия с базой данных для получения нужных данных или выполнения нужных действий.

Преимущества реляционных баз данных

Реляционные базы данных обладают рядом неоспоримых достоинств по сравнению с предшествующими базами данных:

Реляционные базы данных скрывают физическую структуру данных от пользователей и приложений, позволяя им оперировать логическими объектами, вместо непосредственно работы с указателями или иерархическими структурами. Данные в реляционной базе хранятся в виде таблиц с заранее определенной структурой. Это позволяет легко организовать и хранить большие объемы информации в четко структурированном виде, оптимальном для управления и обработки. Универсальная модель данных позволяет создавать и изменять структуру базы данных без необходимости полного переписывания приложения. Например, можно легко добавлять новые таблицы или изменять существующие столбцы без значительного вмешательства в код приложений. Это обеспечивает большую гибкость при разработке и обновлении программных систем.

Язык SQL стандартизирован. Стандарт языка обеспечивает единый язык запросов к реляционным базам данных независимо от того, в рамках

какой СУБД они разрабатывались. Это способствует широкому распространению реляционных баз данных, позволяет разработчикам и администраторам легко переходить от разработки одной реляционной базы данных к другой. Разработчики могут использовать те же навыки и знания SQL для работы с различными системами управления базами данных. Стандартизация SQL также способствует переносимости кода и приложений между различными операционными системами и различными аппаратными платформами.

Важнейшим преимуществом реляционных БД являются их транзакционные свойства. Под транзакцией в контексте баз данных понимается серия из одной или нескольких операций, выполняемых как единая атомарная единица работы. Как правило, это операции добавления, изменения или удаления данных, операции изменения структуры БД. Оформление запросов в единую транзакцию на этапе программирования означает, что либо все ее операции завершатся успешно, либо не реализуется ни одна из них. Транзакции используются для обеспечения согласованности и целостности данных, гарантируя, что база данных останется согласованной даже в случае системных сбоев или ошибок.

ACID-транзакции (Atomicity, Consistency, Isolation, Durability) поддерживаются на уровне СУБД, что гарантирует целостность данных даже при параллельных запросах к одной БД со стороны нескольких пользователей. Для более ясного понимания рассмотрим каждый из этих аспектов.

Атомарность (Atomicity). Транзакция считается атомарной в том смысле, что СУБД не должна позволять обращаться к изменениям, внесенным в БД в рамках транзакции, пока эта транзакция не завершится полностью. Иными словами, никакие изменения, примененные в ходе транзакции, не должны быть видны другим операциям до успешного или неуспешного (откат) завершения транзакции.

Согласованность (Consistency): Транзакция должна переводить базу данных из одного согласованного состояния в другое согласованное состояние. Это гарантирует, что выполнение транзакции не приведет к нарушению целостности данных или других бизнес-правил. Согласованность в контексте баз данных означает, что все правила и ограничения, установленные для базы данных, должны по-прежнему соблюдаться после выполнения транзакции. Пример согласованности хорошо иллюстрируется на примере банковской системы в случае перевод денег с одного счета на другой и обновления информации об общей сумме на счетах.

1. Допустим, у нас есть два счета:

- счет А с балансом 5 000 рублей;
- счет В с балансом 8 000 рублей.

2. Пользователь хочет перевести 2 000 рублей со счета А на счет В. В этот момент начинается транзакция.

3. В рамках транзакции происходят следующие действия:

- со счета А списывается 2 000 рублей;
- на счет В зачисляется 2 000 рублей.

4. После выполнения всех операций база данных должна оставаться в согласованном состоянии:

- сумма на счете А должна быть 3 000 рублей (5 000 – 2 000);
- сумма на счете В должна быть 10 000 рублей (8 000 + 2 000)

Изолированность (Isolation): означает, что когда несколько транзакций выполняются одновременно, каждая из них должна работать так, как будто остальные транзакции вообще не существуют. Другими словами, изменения, внесенные одной транзакцией, не должны быть видимы другим транзакциям до тех пор, пока эта транзакция не завершится.

Долговечность (Durability): означает, что результаты успешно завершенной транзакции должны быть постоянными даже в случае сбоя системы, отказа оборудования или других неполадок. То есть данные, которые были сохранены после успешного завершения транзакции, должны

оставаться в базе данных и быть доступными даже в случае нештатного завершения работы системы.

Таким образом, ACID-транзакции обеспечивают надежное выполнение операций с данными в базе данных, гарантируя их целостность и непротиворечивость. А механизмы для поддержания целостности данных позволяют разработчикам сосредоточиться на более важных аспектах приложения, таких как бизнес-логика и пользовательский интерфейс, не тратя время на написание и отладку сложных проверок после добавления, изменения или удаления данных.

Развитые инструменты администрирования облегчают управление реляционными базами данных, включая мониторинг производительности, резервное копирование и восстановление данных, а также управление доступом для отдельных пользователей и групп пользователей в зависимости от их ролей и задач.

Реляционные базы данных хорошо интегрированы с множеством приложений и инструментов, что делает их универсальным выбором для широкого спектра задач, включая аналитическую обработку OLAP (ONLINE ANALYTICAL PROCESSING), системы оперативной обработки транзакций OLTP (ONLINE TRANSACTION PROCESSING), веб-разработку, аналитику данных и многое другое.

Физическая реализация реляционной модели

Если говорить о соотношении таблиц реляционной модели с хранением в файлах, то таблицы могут быть физически реализованы в виде файлов на диске. Каждая таблица может быть представлена отдельным файлом, в котором хранятся данные в определенном формате, таком как CSV, XML, JSON или в специализированных бинарных форматах. Размещение каждой таблицы базы данных в отдельном файле обеспечивает изоляцию данных и упрощает управление доступом к данным. Каждый файл при этом управляем

отдельно как со стороны СУБД, так и со стороны операционной системы, что облегчает операции по сжатию, восстановлению и резервному копированию. При работе с отдельными файлами для каждой таблицы легче оптимизировать запросы.

Однако в реляционных базах данных может быть реализована и другая концепция, которая называется «мульти-таблицы» или «таблицы в одном файле». Это означает, что несколько таблиц могут быть сохранены в одном файле. Если каждая таблица хранится в своем собственном файле, то управление пространством на диске может быть более эффективным. В некоторых случаях, особенно в небольших приложениях или для временного хранения данных, использование одного файла для нескольких таблиц может быть удобным, поскольку позволяет уменьшить количество файлов в системе и упростить управление базой данных и уменьшить вероятность потери данных или ошибок при копировании или перемещении файлов. Кроме того, это может ускорить резервное копирование и восстановление данных, так как все объекты хранятся в одном файле. Примерами СУБД, использующих мульти-таблицы, являются SQLite и Assecc. В Access файл базы данных (.mdb или .accdb) может содержать не только таблицы, но и другие объекты базы данных, такие как запросы, формы, отчеты и модули. Однако следует помнить, что использование мультитабличной СУБД при работе с большими объемами данных может привести к неоправданному увеличению размера файла базы данных, что может сказаться на производительности и управлении данными. Кроме того, может быть затруднен доступ к разным объектам базы данных из-за конкуренции за ресурсы, если множество пользователей одновременно обращаются к этому файлу базы данных.

Ниже представлен список восьми наиболее популярных реляционных СУБД в соответствии с ежемесячно обновляемым рейтингом популярного IT-сервиса https://db-engines.com/ на март 2024 года. Полный список включает 397 чисто реляционных и реляционных+мультимодельных СУБД.

1. Oracle Database – коммерческая реляционная СУБД, предназначенная для широкого спектра приложений и бизнес-процессов. Имеет высокую производительность, масштабируемость и надежность.

2. MySQL – открытая реляционная СУБД от компании Oracle, широко используемая в веб-разработке и других областях. Обладает хорошей производительностью, простотой использования и надежностью.

3. Microsoft SQL Server – коммерческая реляционная СУБД от Microsoft, часто используемая в корпоративных средах. Предлагает широкий спектр функций, интеграцию с другими продуктами Microsoft и хорошую поддержку.

4. PostgreSQL – мощная открытая реляционная СУБД, известная своей расширяемостью и возможностями для обработки сложных запросов. Поддерживает множество расширений, имеет высокую степень соответствия стандартам SQL.

5. IBM Db2 – коммерческая реляционная СУБД от IBM, предназначенная для корпоративных приложений. Обладает высокой производительностью, масштабируемостью и поддержкой аналитики.

6. Snowflake – облачный сервис хранения данных для структурированных и полуструктурированных данных. Предлагает гибкую масштабируемость, удобное управление данными и возможности аналитики в облаке.

7. SQLite – легковесная встраиваемая СУБД, часто используемая в мобильных и встроенных приложениях. Имеет небольшой размер, простоту использования и хорошую производительность.

8. Microsoft Access – система управления базами данных, которая сочетает в себе ядро реляционной базы данных Access с графическим пользовательским интерфейсом для манипулирования данными и выполнения запросов. Часто используется для небольших проектов и персональных баз данных.

Диаграмма на рисунке 6 показывает популярность реляционных и нереляционных СУБД. Она рассчитывается с учетом популярности всех известных систем по таким критериям как: количество упоминаний системы на веб-сайтах, общий интерес к системе, частота технических обсуждений системы, количество вакансий, в которых упоминается система, количество профилей в профессиональных сетях, в которых упоминается система, актуальность в социальной сети Twitter.

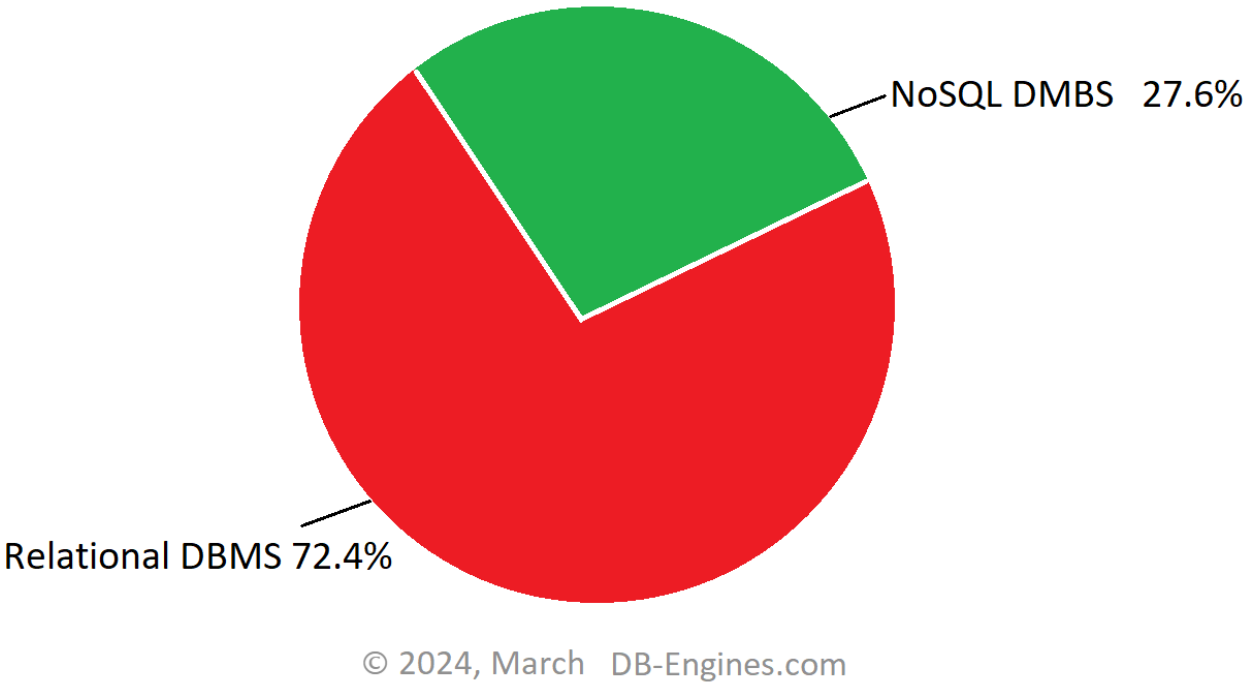

Рисунок 6 – Рейтинговые баллы реляционных и NoSQL-СУБД

Развитие реляционных баз данных. NewSQL

С развитием технологий и появлением новых типов данных, таких как геоданные, временны́е ряды, JSON и XML, реляционные СУБД расширяют свои возможности для эффективного хранения и обработки таких данных. С увеличением объемов данных и нагрузки становится все более важным обеспечение масштабируемости реляционных баз данных. Поэтому Реляционные СУБД успешно интегрируются с другими технологиями и

платформами, такими как облачные сервисы, аналитические инструменты и среды разработки. Разработчики постоянно работают над улучшением производительности реляционных баз данных, включая оптимизацию запросов, индексацию данных, кэширование и другие методы.

Значимым направлением в развитии реляционных баз стала их интеграция с технологиями NoSQL, что привело к появлению концепции NewSQL (новый SQL). Нереляционные базы данных обладают определенными преимуществами, такими как ещё бóльшая гибкость модели данных, горизонтальное масштабирование и высокая производительность при работе с некоторыми видами данных. Поэтому объединение реляционной модели с NoSQL-моделями, способными работать в распределенных средах[3] позволяет разработчикам создавать гибридные системы, такие как крупные веб-сервисы, облачные приложения и аналитические системы, работающие с большими объемами данных и высокими нагрузками. NewSQL базы данных сохраняют реляционную модель данных и совместимость с языком SQL, что обеспечивает привычный и понятный способ организации и хранения данных. Сохраняется удобство использования и легкость перехода для разработчиков, которые знакомы с структурой реляционных баз данных. NewSQL-СУБД сохраняют транзакционные свойства традиционных реляционных баз данных. Они поддерживают ACID-транзакции, что гарантирует целостность данных даже при параллельных операциях.

В то же время, в NewSQL реализуются новые архитектурные решения и механизмы оптимизации запросов, заимствованные у NoSQL, чтобы эффективно обрабатывать большие объемы данных при обеспечении надежности и корректности данных. Они могут масштабироваться на несколько узлов (нод) и серверов для обработки большого количества запросов и операций.

[3] Т.е. на множестве компьютеров-серверов, объединённых в единую высокоскоростную сеть.

Существуют два основных подхода к NewSQL: создание новых систему управления данными с нуля и модификация существующих СУБД для соответствия требованиям NewSQL. Примерами новых систем являются: Google Spanner, CockroachDB, TiDB, MemSQ, VoltDB и др. К их преимуществам следует отнести: гибкость, инновации и оптимизацию производительности. Новые системы управления базами данных могут быть спроектированы с учетом современных требований к масштабируемости и производительности. Разработчики могут внедрять совершенно новые концепции и методы работы с данными, что позволяет создавать более эффективные и мощные решения. При создании СУБД с нуля можно сразу задействовать оптимизации и архитектурные решения, направленные на улучшение производительности и масштабируемости. Примерами усовершенствованных, уже имеющихся и постоянно развивающихся СУБД **с** поддержкой автоматического масштабирования и репликации[4] данных являются:

- MySQL Cluster
- PostgreSQL с версии 9.1
- Oracle Real Application Clusters (RAC) с версии Oracle 10g и выше
- Microsoft SQL Server с версии SQL Server 2005 и выше
- IBM Db2 с версии Db2 Express-C до Db2 Advanced Enterprise Edition
- Amazon Aurora, совместимая с MySQL и Aurora PostgreSQL.

К преимуществам этого подхода относятся: совместимость, надежность и широкая поддержка со стороны сообществ разработчиков. Пользователи этих СУБД могут обновлять свои программные системы без необходимости переписывания кода и перепроектирования баз данных. Усовершенствование существующих СУБД обычно проводится на практически проверенной и надежной основе.

Выбор между этими подходами зависит от конкретных требований,

[4] Т.е. автоматическое поддержание копий данных на нескольких серверах.