Лекция 5. ГРАФОВЫЕ БД. КОЛОНОЧНЫЕ БД. СИСТЕМЫ «КЛЮЧ-ЗНАЧЕНИЕ»

ХРАНИЛИЩЕ «КЛЮЧ – ЗНАЧЕНИЕ»

Хранилища «Ключ – значение» – это самое основное решение из семейства NoSQL. Этот «новый» тип баз данных не такой уж новый и всегда применялся в основном для приложений, для которых использование реляционных БД было бы непригодно. Низкая производительность, отсутствие стандарта и другие недостатки ограничивали использование первых систем «Ключ – значение» в течение многих лет, но быстрое развитие веба и облачных вычислений после 2010 года привело к их возрождению в рамках более широкого движения NoSQL.

Этот тип БД работает с данными подобно словарю. Здесь нет места ни структуре, ни связям между отдельными записями. В модели «Ключ – значение» нет ничего сложного, так как реализовать её очень просто. После подключения к такому серверу БД приложение может заносить в хранилище ключи и соответствующие им значения (данные), а в последствии получать эти данные по запросу, аргументом которого является ключ.

Ключ может быть синтетическим или автосгенерированным, а значение может быть представлено строкой, JSON, набором битовых данных BLOB, (Binary Large Object, большой двоичный объект) и т.д. Такие базы данных как правило используют хеш-таблицу, в которой находится уникальный ключ и указатель на конкретный объект данных

Приложение может хранить в наборе значений произвольные данные, но некоторые хранилища пар «Ключ – значение» накладывают ограничения на максимальный размер значений. Программное обеспечение хранилища ничего не знает о значениях, которые в нем хранятся. Все сведения о схеме (т.е. о структуре хранимых данных во поле «Значение») поддерживаются и применяются на уровне приложения. В глобальном смысле с точки зрения СУБД эти значения по существу являются большими двоичными объектами, которые хранилище извлекает и сохраняет по соответствующему ключу (рисунок 5.1).

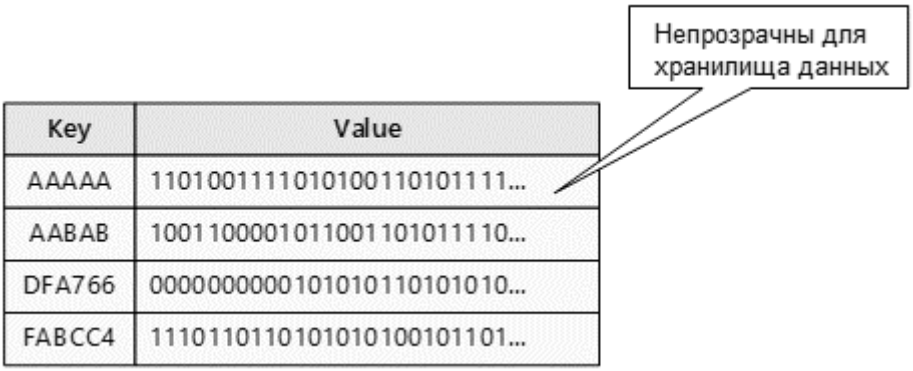

Рисунок 5.1. Пример хранимых пар «ключ – значение»

В некоторых реализациях хранилищ значения могут быть только строковыми. В других реализациях атрибуты могут иметь простые типы данных, которые отражают типы, использующиеся в программировании: целые числа, массивы строк и списки. Но главное, что межу разными значениями и внутри одного значения никакие отношения явно не определены.

Существует понятие *блока* (bucket) – *логической группы ключей*, однако в них данные физически не группируются. В разных блоках могут присутствовать идентичные ключи. Чтобы прочитать значение, необходимо знать, как ключ, так и блок, поскольку на самом деле ключ является хешем (блок + ключ).

Если рассматривать теорему CAP применительно к хранилищам «Ключ – значение», то становится довольно очевидно, что такие хранилища хороши в плане доступности (Availability) и устойчивости к разделению (Partition tolerance), но явно проигрывают в согласованности данных (Consistency).

Операции СУБД «Ключ-значение»

Большинство хранилищ пар «Ключ – значение» поддерживают только самые простые операции запроса, вставки и удаления.

Примеры операций чтения и записи:

Get(key) возвращает значение, связанное с аргументом key;

Put(key, value) связывает значение value с ключом key;

Multi-get(key1, key2, ..., keyN) возвращает список значений, связанных с переданным ключами key1..keyN;

Delete(key) удаляет запись для ключа key из хранилища.

Чтобы частично или полностью изменить значение, приложение всегда перезаписывает существующее значение целиком. В большинстве реализаций атомарной операцией считается чтение или запись одного значения. Запись больших значений занимает относительно долгое время.

Некоторые реализации предоставляют SQL-подобный синтаксис для задания условий фильтрации. Зачастую можно использовать только базовые операторы сравнений (=, !=, <, >, <= и =>). Вся бизнес-логика и логика для поддержки целостности данных содержится в коде приложений.

Возможности и ограничения БД «Ключ – значение»

Обеспечивают беспрецедентное горизонтальное масштабирование, недостижимое при использовании других типов баз данных за счет простоты модели.

Хранилища пар «Ключ – значение» рассчитаны на приложения, выполняющие простые операции поиска на основе значения ключа, набора ключей или диапазона ключей, но не очень подходят для систем, которым нужно запрашивать и далее присоединять друг к другу данные из нескольких таблиц хранилищ.

Кроме того, хранилища «Ключ – значение» неудобны в сценариях, где могут выполняться запросы или фильтрация по значению, а не только по ключам. Например, с помощью реляционной базы данных можно найти запись, используя предложение WHERE для фильтрации неключевых столбцов, но в хранилищах «Ключ – значение» обычно отсутствует возможность поиска в значениях, а если они есть, потребуется медленный просмотр всех значений.

Модель данных не предоставляет стандартные возможности традиционных баз данных вроде атомарности транзакций или согласованности данных при одновременном выполнении нескольких обращений к нескольким хранилищам. Такие возможности должны предоставляться самим приложением.

При увеличении объёмов данных поддержание уникальных ключей может стать проблемой. Для её решения необходимо как-то усложнять процесс генерации ключей, чтобы они оставались уникальными среди очень большого набора.

Большинство хранилищ данных выполняют обработку запросов и данных на стороне сервера. Иногда эта функция встроена в подсистему хранилища данных. В других случаях функции обработки данных вынесены в отдельный модуль или несколько модулей для обработки и анализа. Хранилища данных также поддерживают разные интерфейсы программного доступа и управления.

Популярные СУБД «Ключ – значение»

Перечень первых двадцати популярных хранилищ «Ключ – значение» по версии DB_Engines (из пятидесяти восьми известных) на декабрь 2021 г. представлен в таблице 5.2. Некоторые хранилища поддерживают и другие NoSQL-модели.

Таблица 5.2 Популярные хранилища «Ключ – значение»

№	СУБД	Тип	Другие поддерживаемые типы	Рейтинг
1.	Redis	Key-value, Multi-model	Document store Graph DBMS Spatial DBMS Search engine Time Series DBMS	173.54
2.	DynamoDB (Amazon)	Key-value, Multi-model	Document store Graph DBMS Wide column store Spatial DBMS	77.63
3.	Azure Cosmos DB (Microsoft)	Key-value, Multi-model	Document store, Graph DBMS, Wide column store, Spatial DBMS	39.71
4.	Memcached	Key-value		26.27
5.	etcd	Key-value		10.96
6.	Hazelcast	Key-value, Multi-model	Document store	9.56
7.	Riak KV	Key-value		6.50
8.	Ignite	Key-value, Multi-model	Relational DBMS	6.43
9.	Ehcache	Key-value		6.10
10.	Aerospike	Key-value, Multi-model	Document store	5.55
11.	ArangoDB	Key-value, Multi-model	Document store Graph DBMS Search engine	4.75
12.	OrientDB	Key-value, Multi-model	Document store Graph DBMS	4.40
13.	Oracle NoSQL	Key-value, Multi-model	Document store Relational DBMS	4.23
14.	ScyllaDB	Multi-model	Wide column store	3.93
15.	RocksDB	Key-value		3.70
16.	InterSystems Caché	Key-value, Multi-model	Object oriented DBMS Relational DBMS Document store	3.15
17.	LevelDB	Key-value		2.94
18.	Oracle Berkeley DB	Key-value, Multi-model	Native XML DBMS	2.94
19.	Infinispan	Key-value		2.81
20.	Oracle Coherence	Key-value		2.64

Redis. Redis расшифровывается как Remote Dictionary Server, – это быстрое хранилище данных типа «ключ-значение» в памяти с открытым исходным кодом. Redis обеспечивает время отклика на уровне долей миллисекунды и позволяет приложениям, работающим в режиме реального времени, выполнять миллионы запросов в секунду. Такие приложения востребованы в сферах игр, рекламных технологий, финансовых сервисов,

здравоохранения и интернета вещей. Все данные Redis хранятся в памяти, что обеспечивает низкую задержку и высокую пропускную способность доступа к данным.

Типы данных Redis включают:

- строки – текстовые или двоичные данные размером до 512 МБ;
- списки – коллекции строк, упорядоченные в порядке добавления;
- множества – неупорядоченные коллекции строк с возможностью пересечения, объединения и сравнения с другими типами множеств;
- сортированные множества – множества, упорядоченные по значению;
- хэш-таблицы – структуры данных для хранения списков полей и значений;
- битовые массивы – тип данных, который дает возможность выполнять операции на уровне битов;
- структуры HyperLogLog – вероятностные структуры данных, служащие для оценки количества уникальных элементов в наборе данных;
- потоки – очереди сообщений со структурой журналов данных;
- пространственные данные – записи карт на основе долготы/широты, «поблизости».

Благодаря этому Redis позволяет писать меньше строк для хранения, использования данных и организации доступа к данным в приложениях. Разработчикам под Redis доступны более ста клиентов с открытым исходным кодом. Поддерживаемые языки программирования включают Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go и многие другие.

Amazon DynamoDB. Amazon DynamoDB – это нереляционная база данных, обеспечивающая надежную производительность при любом масштабе. База данных является полностью управляемой и работает в нескольких регионах с несколькими ведущими серверами. Она обеспечивает устойчивую задержку в пределах нескольких миллисекунд и обладает встроенными средствами безопасности, резервного копирования и восстановления, а также кэширования в памяти. В DynamoDB элемент состоит из первичного или сложного ключа и переменного количества атрибутов. Явно заданных ограничений по количеству атрибутов, связанных с отдельным элементом, не существует, однако суммарный размер элемента, включая все имена и значения атрибутов, не должен превышать 400 кб. Таблица представляет собой совокупность элементов данных, подобную совокупности строк в таблице реляционной базы данных. Каждая таблица может содержать бесконечное количество элементов данных.

Примеры использования

Хорошо использовать такие БД для хранения данных сессий в приложениях, работающих в масштабе всего Интернета, а также для управления такими данными; для организации счётчиков посещений; для хранения метаданных профилей пользователей; для хранения историй и счётчиков просмотров; для поддержки аналитической обработки данных в режиме реального времени, например, для аналитики в социальных сетях, рекламного таргетинга, персонализации контента и интернета вещей и т.д.

Организация высокоскоростного кэша в памяти программной системы. Создание кэша на основе хранилища «Ключ – значение» позволит уменьшить задержку доступа, существенно увеличить пропускную способность, снизить нагрузку на реляционную базу данных или базу данных NoSQL, снизить нагрузку на само приложение.

Хранилище сессий. Основанное на сессиях приложение (например, интернет-приложение) запускает сессию, когда пользователь входит в систему. Оно активно до тех пор, пока пользователь не выйдет из системы или не истечет время сессии. В течение этого периода приложение хранит все связанные с сессией данные либо в основной памяти, либо в базе данных. Данные сессии могут включать информацию профиля пользователя, сообщения, индивидуальные данные и темы, рекомендации, целевые рекламные кампании и скидки. Каждая сессия пользователя имеет уникальный идентификатор. Данные сессий всегда запрашиваются только по первичному ключу, поэтому для их хранения отлично подходит быстрое хранилище «Ключ – значение». В целом базы данных на основе пар «ключ-значение» могут снижать накладные расходы в расчете на страницу по сравнению с реляционными базами данных.

Корзина интернет-магазина. Во время праздничного сезона покупок сайт интернет-магазина может получать миллиарды заказов за считаные секунды. Используя базы данных на основе пар «Ключ – значение», можно обеспечить необходимое масштабирование при существенном увеличении объемов данных и чрезвычайно интенсивных изменениях состояния. Такие базы данных позволяют одновременно обслуживать миллионы пользователей благодаря распределенным обработке и хранению данных. Базы данных на основе пар также обладают встроенной избыточностью, что позволяет справляться с потерей узлов хранилища.

Перспективы хранилищ «Ключ – значение»

Хранилища данных, включенные в категорию «Ключ – значение», не обязательно предоставляют одинаковый набор возможностей.

Многие из таких СУБД стараются сохранить простоту чистого интерфейса хранилища «Ключ – значение», но с некоторыми дополнительными функциями, добавленными для удовлетворения общих требований. Другие ориентированы на расширение базовой модели данных.

Перспективным развитием рассмотренного типа хранилищ, по нашему мнению, должна стать некая золотая середина функциональности, которая сохранит преимущества минимального набора функций, не требуя при этом множества усилий со стороны разработчика приложения.

ГРАФОВЫЕ БД И СУБД

Графовая БД, также называемая графо-ориентированной БД использует узлы для хранения сущностей данных и ребра для хранения взаимосвязей между сущностями (рисунок 5.2).

Рисунок 5.2. Элементы графовой модели данных

Графовые СУБД – это СУБД, которые используют графовую модель для описания и манипуляции данными в базе данных. Они относятся к направлению NoSQL, которое объединяет множество различных подходов к моделированию данных, отрицающих реляционную модель данных. Однако, в отличии от других моделей, графовые СУБД, как и базы данных SQL, поддерживают ACID-транзакции. ACID – atomic, consistent, isolated durable – атомарные, согласованные, изолированные и долговечные.

Система управления графовыми базами данных поддерживает методы создания (Create), чтения (Read), изменения (Update) и удаления (Delete) (CRUD), основан-ные на графовой модели данных. Графовые базы данных, как прави-ло, поддерживают систему транзакций реального времени (OLTP). Соответственно, они оптимизированы для выполнения транзакций и спроектированы с учетом транзакционной целостности и опера-тивности.

Первая графовая СУБД Neo4j была создана в 2007 году. В настоящее время созданы и успешно функционируют десятки разнообразных подклассов (в зависимости от реализации узлов и рёбер) графовых СУБД. Обычно графовые СУБД не предоставляют индексы для всех узлов, прямой доступ к

узлам на основе значений атрибутов в этих случаях невозможен. Прежде всего графовые базы данных предназначены для хранения взаимосвязей и навигации в них. Взаимосвязи в графовых базах данных являются объектами высшего порядка, в этом заключается основная ценность этих баз данных.

Начиная с версии 2017, графовую модель данных стала поддерживать СУБД MS SQL Server.

Графовая модель

Модель хранения информации в виде графов и гиперграфов сложилась в 1990-2000 годах, хотя использование графов в виде модели представления данных сложилось ещё с 1980-х годов. Графовая модель данных на логическом уровне представляет собой направленный граф, состоящий из узлов и ребер. Узлы соответствуют объектам базы данных, а ребра – связям между этими объектами. И узлы, и ребра могут обладать свойствами. Кроме свойств ребра имеют тип, определяющий характер связи. Ребро всегда имеет начальный узел, конечный узел, тип и направление. Ребра могут описывать самые разнообразные взаимосвязи, например, отношения иерархии типа «родитель-потомок», действия, права владения и т. п. Ограничения на количество и тип взаимосвязей, которые может иметь узел, отсутствуют.

Обход графа в графовой базе данных можно выполнять либо по определенным типам ребер, либо по всему графу. Обход соединений или взаимосвязей в графовых базах данных выполняется очень быстро, поскольку взаимосвязи между узлами не вычисляются во время выполнения запроса, а хранятся в базе данных.

Графовая модель хорошо отражает семантику предметной области с многочисленными связями. Эту модель данных обычно рассматривают как обобщение сетевой модели данных или RDF-модели (Resource Description Framework – это разработанная консорциумом Всемирной паутины модель для представления данных, в особенности — метаданных).

Ниже приведен пример графа социальной сети. Имея данные о людях (узлы) и взаимосвязях между ними (ребра), можно узнать, кто является «друзьями друзей» конкретного человека, например, пользователя по имени Oleg (рисунок 5.3).

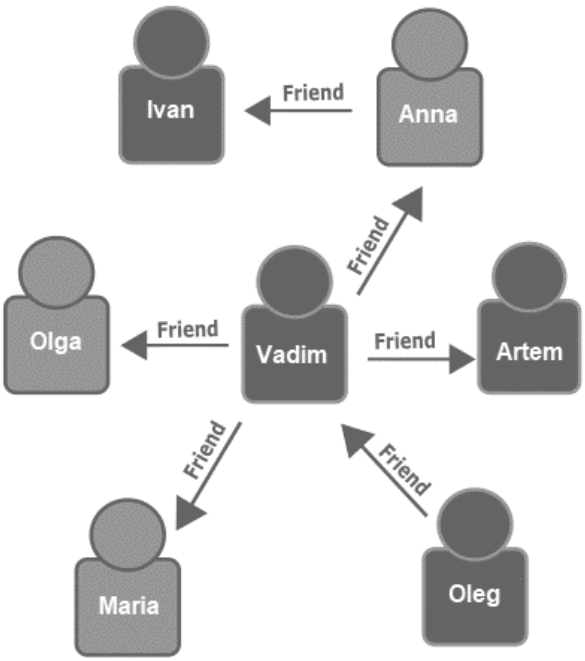

Рисунок 5.3. Фрагмент графа социальной сети с типом связи «Friend»

Еще один пример графовой модели с одиночными связями, которая отображает данные персонала организации. Сущностями здесь являются сотрудники и отделы, а рёбра определяют отношения подчинения и отдел, в котором работает каждый сотрудник. Стрелки на ребрах этого графа показывают направление связей. Эта структура позволяет легко выполнять такие запросы, как «Найти всех сотрудников, которые напрямую или косвенно взаимодействуют с Инессой» или "Кто работает в одном отделе с Владимиром?" Для больших диаграмм с большим количеством сущностей и связей можно быстро выполнять сложный анализ взаимосвязей. Язык запросов графовых БД предназначен для эффективного обхода сети с целью поиска нужных связей.

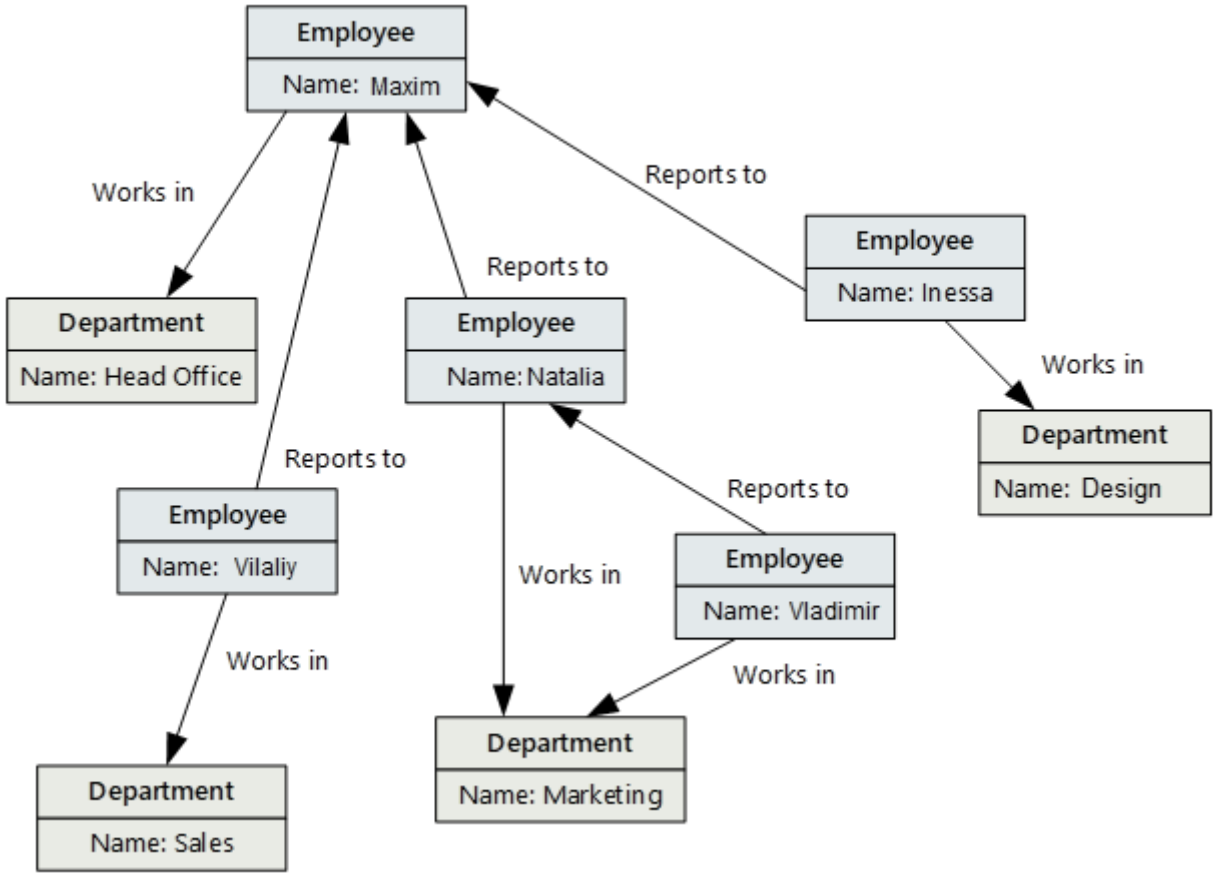

Рисунок 5.4. Фрагмент графа организации с сущностями «Отдел» и «Служащий»

Социальные данные имеют множество связей друг с другом и ох очень сложно представить в виде взаимосвязанных таблиц реляционной БД или отдельных «документов» документо-ориентированной БД. Пример такого графа представлен на рисунке 5.5. Когда пользователь попадает в социальную сеть, есть только одна важная часть страницы – его лента активности. Запрос ленты активности получает все посты от друзей пользователя, отсортированных по убыванию даты. Каждый пост содержит вложения, такие как фотографии, лайки, репосты и комментарии. Друзья тоже имеют посты, посты имеют комментарии и лайки, каждый из которых имеет связан с одним комментатором или лайкером. Точно также комментаторы и лайкеры также могут быть пользователями. Для каждого входящего в социальную сеть пользователя необходимо получить всю структуру разом, как только он войдет откроет сессию. В реляционной СУБД это было бы соединение семи таблиц, чтобы вытащить все данные. Обновление данных пользователя означает обход всех лент активности, с целью изменения данных во всех местах, где они хранятся. Этот процесс очень сильно подвержен ошибкам, и часто приводит к несогласованности данных, особенно в случае удаления информации. Концептуально, социальные данные более напоминают граф, чем набор таблиц.

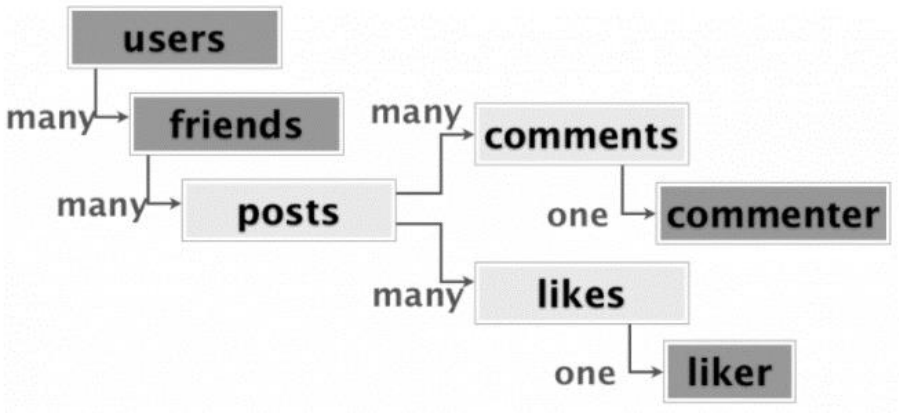

Рисунок 5.5. Пример графа, описывающего социальные связи

На самом деле в мире очень мало понятий, которые естественно могут моделироваться в виде нормированных таблиц. К тому же, наше время характеризуется высоким ростом связности данных. При проектировании приложений нередко появляются новые требования к структуре данных, и изначальная реляционная или альтернативная ей документо-ориентированная модель может оказаться совсем не эффективной. Например, добавление новых связей делает неприемлемой документо-ориентированную БД, а рост количества соединений (JOIN-ов) катастрофически снижает производительность реляционной БД. В этом случае графовая модель оказываются наиболее универсальным решением, позволяющим подстраховаться на случай изменения требований и расширения функционала в будущем. В неё легко добавляются дополнительные связи, достаточно просто реализуется процесс изменения иерархической структуры.

Другой пример – постепенное превращение Интернета из большой «информационной свалки» в большую базу знаний. Чтобы превратить информацию в знания, в первую очередь необходимо так или иначе связывать блоки информации между собой. Так, аналитическое агентство Gartner в своем исследовании «Hype Cycle for Emerging Technologies» начиная с 2018 года указывает на «Knowledge Graphs» как на восходящий тренд в семействе развивающихся технологий.

Наконец, укрупнение и интеграция информационных сервисов, характерные для современного этапа развития корпоративных информационных систем, сопровождаются объединением разнородных баз данных в единые комплексы. Это затратно реализовать на основе реляционной модели, но практически безболезненно может выполнить с использованием графовой модели.

Разновидности графовых моделей, язык SPARQL

Существует две основных разновидности графовой модели данных: Property Graph и RDF-граф (RDF — Resource Description Framework).

Property Graph – это ориентированный граф, в котором вершины соответствуют «записям», а ребра — «связям» между ними. Дополнительно и ребра, и связи могут быть снабжены скалярными атрибутами. Наиболее известной СУБД, работающей с моделью Property Graph, является Neo4j со своим оригинальным декларативным языком запросов Cypher. В основном, СУБД, работающие с Property Graph, имеют проприетарные интерфейсы и несовместимы друг с другом.

Модель данных RDF еще раньше развивалась в недрах науки искусственного интеллекта как один из способов представления знаний. После 2001 года, когда в журнале Scientific American была опубликована статья автора идеи всемирной паутины (World Wide Web) Тима Бернерса Ли, в Internet-сообществе стал лавинообразно нарастать интерес к графовым базам, в частности, RDF. В 2004 г. RDF принят как стандарт комитета W3C (World Wide Web Consortium).

RDF – это формат задания графа, представленный в виде троек «субъект – предикат – объект» (рисунок 5.6).

Рисунок 5.6. Базовая триада RDF

Пример на рисунке 5.7 показывает десять троек, определяющих человека по имени Петр Иванов как автора некоторой статьи в журнале «Вопросы биологии». RDF-тройки по несложным правилам логически могут быть «упакованы» в граф, изображенный на рисунке 5.8.

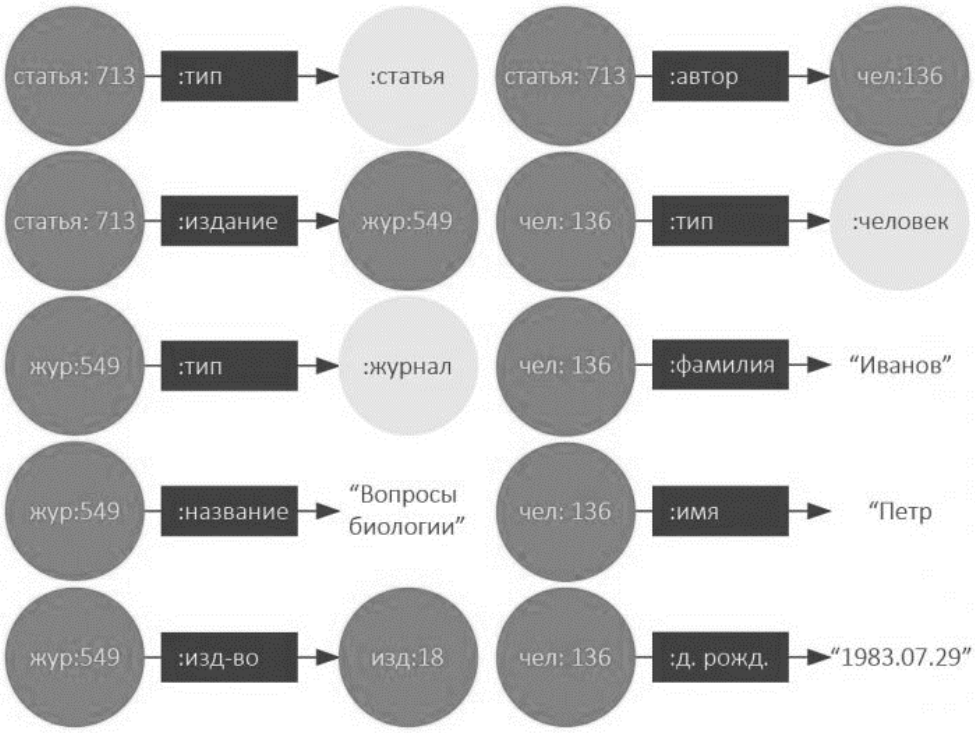

Рисунок 5.7 Отображение сведений об авторе статьи на основе RDF-троек

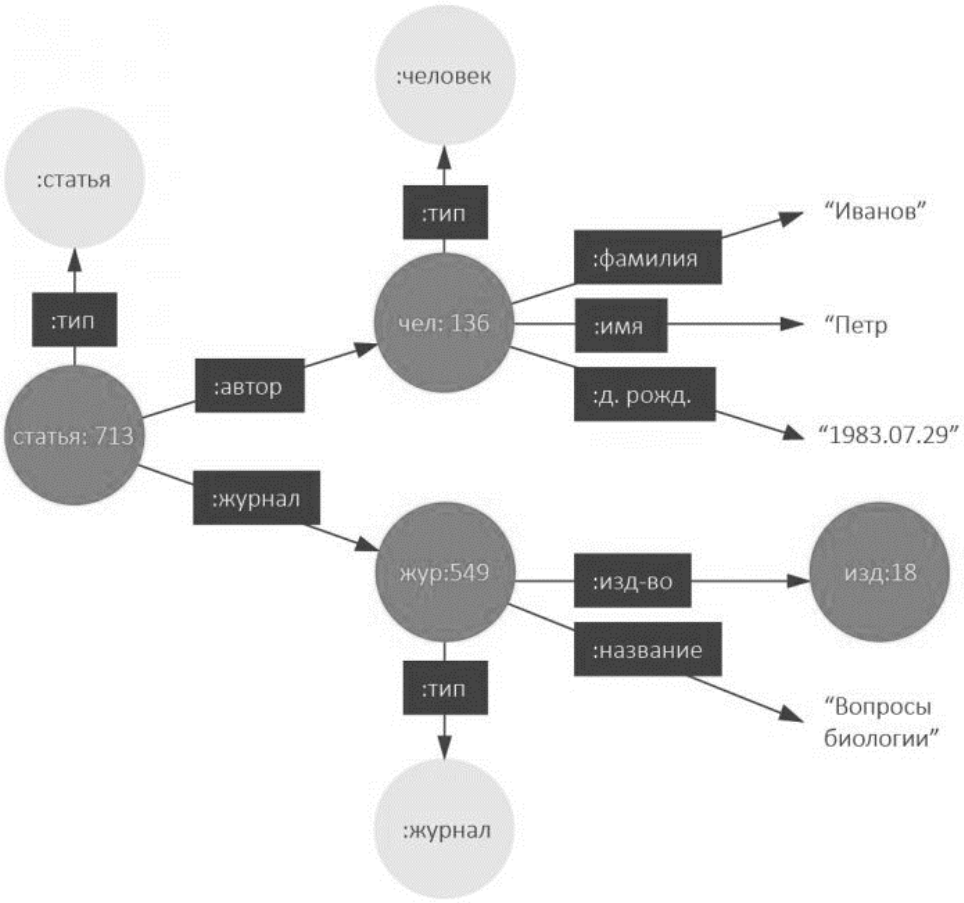

Рисунок 5.8. Упаковка в граф сведений об авторе статьи

На графе, заданном таким образом, можно делать запросы с помощью специальных языков. Наиболее известный из них – язык, принятый в качестве стандарта W3C –SPARQL. Его синтаксис похож на SQL, язык SPARQL, как и SQL, имеет декларативную основу, поддерживает не только операцию чтения, но и операции манипулирования данными (вставить, удалить и др.).

Например, на полном графе, фрагмент которого представлен на рисунке 5.8, можно с помощью SPARQL запроса найти всех авторов, печатавшихся в издательстве «Вопросы биологии», или, например, найти в каких издательствах печатался автор Петр Иванов, родившийся 2 июля 1980 г. и т.п.

Благодаря усилиям консорциума W3C модель RDF удалось сделать гораздо более стандартизованной, чем это было достигнуто для реляционной модели.

1. Язык SPARQL гораздо лучше стандартизован, чем SQL. Запросы на SPARQL можно смело посылать разным графовым RDF СУБД, содержащим одинаковые данные, результат будет одинаковым.
2. RDF стандартизует также идентификацию информации (концепция URI – Universal Resource Identifier).
3. Стандартизован протокол взаимодействия компонентов (HTTP), точка доступа SPARQL, и т.д.

Что особенно важно, графовая RDF-модель гораздо легче проектируется, управляется и модифицируется, чем аналогичная реляционная модель данных. Поиск сложных связей на этой модели также происходит с гораздо большей эффективностью, поскольку все связи прямые, нет необходимости проводить операцию объединения через внешние ключи.

По оценкам специалистов переход от SQL-баз данных к RDF-системам обещает такой же технологический скачок, как переход от самых первых СУБД к SQL.

Механизмы вычисления графов

Механизмы вычисления графов позволяют выполнять глобальные графовые вычислительные алгоритмы для больших наборов дан-ных. Они предназначены для решения таких задач, как идентифи-кация кластеров данных или получение ответов на такие вопросы, как: «Сколько всего взаимосвязей, сколько их в среднем, полна ли социальная сеть?»

Из-за своей направленности на глобальные запросы механизмы вычисления графов, как правило, оптимизированы для сканирования и пакетной обработки больших объемов информации, и в этом отно-шении они похожи на другие технологии пакетного анализа, такие как интеллектуальный анализ данных (data mining) или аналитиче-ская обработка в реальном времени (OLAP), используемые в реля-ционном мире. Некоторые механизмы вычисления включают в себя и средства хранения графов, а другие (большинство) заботятся толь-ко об обработке данных, получаемых из

внешнего источника, а затем возвращают результаты для сохранения в другом месте.

На рисунке 5.9 представлена обобщенная схема развертывания механизма вычисления графов. Она включает в себя систему записи (System of Record, SOR) базы данных со свойствами OLTP (например, MySQL, Oracle или Neo4j), которая обслуживает запросы и отвечает на запросы, поступающие от приложений, а в конечном счете от пользователей. Периодически задания на извлечение, преобразование и загрузку данных (Extract, Transform, Load, ETL) перемещают данные из системы записи базы данных в механизм вычисления графов для выполнения автономных запросов и анализа.

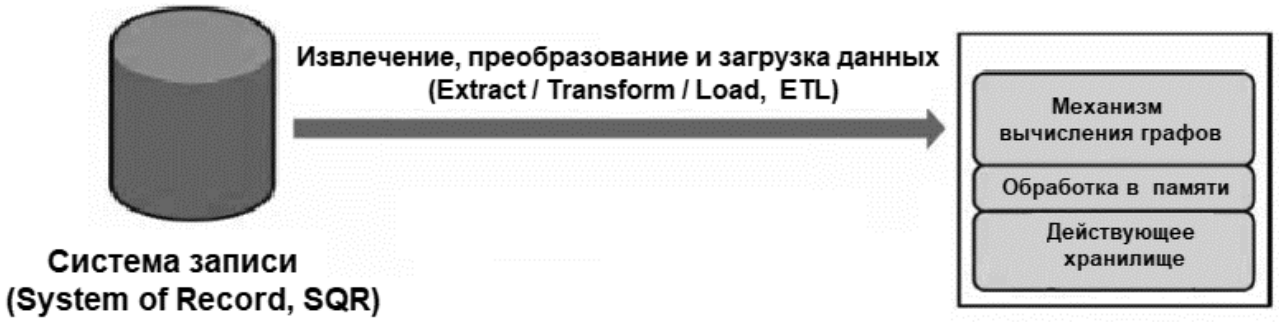

Рисунок 5.9. Схема типичной среды, выполняющей расчет графов

Существуют разные типы механизмов вычисления графов. Наиболее известными из них являются одномашинные (in-memory/single machin), которые хранят информацию непосредственно в оперативной памяти, и распределенные, такие как Pegasus и Giraph.

Достоинства и недостатки графовых БД

Универсальность – отличительная черта графовых БД. В них можно хранить и реляционные, и документальные, и сложные семантические данные.

Графовая модель может меняться и модифицироваться в процессе развития приложения без изменения архитектуры и исходных запросов. А значит не нужно будет переписывать код или переходить к другой модели.

В случае необходимости данные из графового хранилища могут быть легко преобразованы в другую модель.

Для задач с естественной графовой структурой данных графовые СУБД могут существенно превосходить реляционные по производительности, а также иметь преимущества в наглядности представления и простоте внесения изменений в схему базы данных.

Гибкое графическое представление идеально подходит для решения проблем масштабируемости.

Важным ограничением является то, что в данный момент практически не существует графовых баз данных, которые бы хорошо работали в параллельных архитектурах.

При незначительном количестве связей и больших объемах данных графовые БД демонстрируют значительно более низкую производительность, чем классические реляционные решения.

Популярные графовые СУБД

Перечень первых двадцати популярных графовых СУБД (из тридцати шести известных) по версии DB_Engines на декабрь 2021 г. представлен в таблице 5.3. Некоторые СУБД поддерживают и другие NoSQL-модели.

Таблица 5.3. Популярные графовые СУБД

№	СУБД	Тип	Поддерживаемые типы	Рейтинг
1.	Neo4j	Graph		58.03
2.	Microsoft Azure Cosmos DB	Graph Multi-model	Document store Key-value store Wide column store Spatial DBMS	39.71
3.	Virtuoso	Graph Multi-model	Document store, Native XML DBMS Relational DBMS RDF store Search engine Spatial DBMS	5.07
4.	ArangoDB	Graph Multi-model	Document store Native XML DBMS Relational DBMS RDF store Search engine Spatial DBMS	4.75
5.	OrientDB	Graph Multi-model	Document store, Key-value store Object-oriented model	4.40
6.	GraphDB	Graph RDF store		2.88
7.	Amazon Neptune	Graph RDF store		2.56
8.	JanusGraph	Graph		2.41
9.	TigerGraph	Graph		2.00
10.	Stardog	Graph RDF store		1.93
11.	Dgraph	Graph		1.58
12.	Fauna	Graph Multi-model	Document store Relational DBMS Time Series DBMS	1.40
13.	Giraph	Graph	Document store	1.35

№	СУБД	Тип	Поддерживаемые типы	Рейтинг
		Multi-model	RDF store Spatial DBMS	
14.	AllegroGraph	Graph Multi-model	Document store RDF store Spatial DBMS	1.23
15.	Nebula Graph	Graph		1.14
16.	Blazegraph	Graph RDF store		0.96
17.	Graph Engine	Graph Multi-model	Key-value store	0.85
18.	TypeDB	Graph Multi-model	Graph DBMS Relational DBMS	0.78
19.	InfiniteGraph	Graph		0.46
20.	Memgraph	Graph		0.37

Neo4J. Эта графовая СУБД разработана компанией Neo Technology в 2009 г. Самая последняя стабильная на текущий момент версия – 1.9. Она доступна в трех вариантах: Community, Advanced и Enterprise. Версия Community распространяется по лицензии AGPL v3, для коммерческих же проектов необходимо приобретать версию Advanced, включающую дополнительные возможности мониторинга состояния базы. Версия Enterprise, кроме того, поддерживает резервное копирование и масштабируемость. На сегодняшний день Neo4J является наиболее популярной графовой БД, в первую очередь ввиду того, что бесплатная версия предлагает все необходимые инструменты для полноценной работы с графами.

Neo4J может устойчиво хранить данные на жестком диске. Следовательно, объем хранящейся информации ограничен лишь объемами жесткого диска. Для достижения максимальной производительности в Neo4J существует два типа кэширования: файловый кэш (file buffer cache) и объектный кэш (object cache). Первый кэширует данные с жесткого диска, целью чего является увеличение скорости чтения/записи на жесткий диск. Второй кэш хранит в себе различные объекты графа: вершины, ребра и свойства в специальном оптимизированном формате для увеличения производительности обходов графа. Neo4J обладает режимом импорта большого объема данных BatchInserter. В этом режиме происходит отключение транзакций в БД, следствием чего является резкое увеличение скорости импорта. Кроме того, в Neo4J поддерживаются алгоритмы обхода графов, в том числе можно сделать обход в ширину до заданного уровня, что как раз и необходимо для решения задачи поиска соседних вершин.

OrientDB. OrientDB первоначально выпущен в 2010 г. OrientDB реализован на Java, поэтому его можно запускать во всех операционных системах с Java JDK ≥ JDK6. Это СУБД с открытым исходным кодом, но коммерческая поддержка также доступна от компании OrientDB. СУБД поддерживает множество языков программирования, включая: .Net, C, C #, C ++, Clojure, Java, JavaScript, JavaScript (Node.js), PHP, Python, Ruby и

Scala.OrientDB. Это мультимодельная система управления базой данных без схемы, поддерживающая графы, документы, и пары «ключ – значение» и объектную модель. Он поддерживает режимы без схемы, с полной схемой и со схемой. Для обхода графа поддерживаются язык обхода графов Gremlin и SQL-запросы.

Примеры использования

Графовые базы данных имеют ряд преимуществ в таких примерах использования, как социальные сети, сервисы рекомендаций и системы выявления мошенничества, когда требуется создавать взаимосвязи между данными и быстро их запрашивать.

Выявление мошенничества. Графовые базы данных позволяют выявлять сложные схемы мошенничества. Анализ взаимосвязей в графовых базах данных дает возможность обрабатывать финансовые операции и операции, связанные с покупками, практически в режиме реального времени. С помощью быстрых запросов к графу можно, например, определить, что потенциальный покупатель использует тот же адрес электронной почты и кредитную карту, которые уже использовались в известном случае мошенничества. Графовые базы данных также позволяют без труда обнаруживать определенные шаблоны взаимосвязей, например, когда несколько человек связаны с одним персональным адресом электронной почты или, когда несколько человек используют один IP-адрес, но проживают по разным физическим адресам.

Сервисы рекомендаций. Графовые базы данных хорошо зарекомендовали себя для рекомендательных приложений. В графе можно хранить взаимосвязи между такими информационными категориями, как интересы покупателя, его друзья и история его покупок. С помощью высокодоступной графовой базы данных можно рекомендовать пользователям товары на основании того, какие товары приобретали другие пользователи, которые интересуются тем же видом спорта и имеют аналогичную историю покупок. Или можно найти людей, у которых есть общий знакомый, но которые еще не знакомы друг с другом, и предложить им подружиться.

Государственный сектор. Управление процессами, документооборот, аналитика, социологические исследования, и т.д. Клиентами могут быть федеральные и региональные министерства, исследовательские организации, и т.д.

Жизненный цикл предприятий. Интеграция и управление жизненным циклом предприятий в системах автоматизации предприятий. Применение графовых технологий дает возможность единообразно хранить и обрабатывать очень разнородную информацию, включая данные систем ERP, EAM, PLM, САПР и т.д. Клиенты – компании, переходящие на ISO 15926 и другие стандарты на основе RDF, например, Bentley, Siemens, OAK, OCK, Росатом, Роснефть, РАО ЕЭС и т.д.

Средства массовой информации. Единая база событий, фактов, компаний, предприятий, журналистов, изданий, статей, сообщений и т.д.

Образование. Построение графа взаимосвязей между терминами, понятиями, программами обучения, задачами, тестами, знаниями учащегося и т.д. для построения адаптивных обучающих систем. Клиенты – учебные заведения, провайдеры систем дистанционного обучения.

Оборона. АСУ вооруженными силами; системы распределения военной информации; системы планирования войсковых операций; системы анализа и поддержки принятия решений на ТВД; средства автоматизации процессов боевого информационного обеспечения и управления войсками; тренажерные системы; и т.д.

Безопасность. Анализ логов, электронной почты, интернет трафика; финансовый анализ; анализ информации о людях из разных источников, включая данные о работе, собственности, доходах; анализ информации о связях и интересах из социальных сетей; поиск связей по аффилированности и т.д. Возможные клиенты – службы безопасности банков и предприятий, холдингов, корпораций и т.д.

Бизнес-аналитика. Построение аналитических систем нового поколения; построение систем Semantic Data Warehouse — нового класса хранилищ, использующего взаимосвязи данных для повышения качества принятия решений. Клиенты – государственный сектор, банки, предприятия, корпорации, страховые компании и т.д.

Геосервисы, геоприложения. Построение взаимосвязей разнородных данных на карте, связанной с людьми, местами и событиями, компаниями, предприятиями и т.д.; решение задач логистики, маршрутизации и т.д. Возможные клиенты –Министерство обороны, МЧС, другие министерства, разработчики социальных приложений и т.д.

Перспективы графовых БД

Активно идет доработка RDF – основного стандарта, согласно которому работают графовые базы данных. Рассуждая по аналогии, можно вспомнить, что именно стандартизация SQL сделала такими популярными реляционные БД.

Совершенствуется язык SPARQL, обладающий обширными возможностями для работы с различными видами запросов.

За счет развития архитектуры производительность графовых БД растет, и, возможно, скоро окажется выше реляционной даже при небольшом количестве связей.

КОЛОНОЧНЫЕ БД И СУБД

Колоночные хранилища или столбчатые хранилища – это тип базы данных NoSQL, в которых данные сгруппированы по столбцам, а не по

строкам. Столбчатые хранилища данных в простейшей форме реализации почти неотличимы от реляционной базы данных, по крайней мере организационно. Настоящее преимущество столбчатого хранилища данных перед традиционным строковым хранилищем (реляционным) заключается в способности денормализованно структурировать разреженные данные, что связано со столбцово-ориентированным методом хранения данных.

Колоночное и строковое хранение данных

Рассмотрим разницу в хранении строковых и колоночных данных на простом примере. Например, в базе данных может храниться такая таблица:

RowID	**EmpID**	**Lastname**	**Firstname**	**Salary**
001	10	Joe	Smith	60000
002	12	Mary	Jones	80000
003	11	Cathy	Johnson	94000
004	22	Bob	Jones	55000

Такое двумерное представление является логической абстракцией. Для хранения на диске данные должны быть сериализованы в ту или иную форму (*сериализация* – процесс перевода структуры данных в последовательность байтов). Распространенный метод хранения табличных данных – сериализация каждой строки данных в форму:

```
001:10,Smith,Joe,60000;
002:12,Jones,Mary,80000;
003:11,Johnson,Cathy,94000;
004:22,Jones,Bob,55000;
```

Когда данные помещаются в таблицу, им присваивается внутренний идентификатор RowID который используется внутри системы для ссылки на данные. На практике обычно используются крупные 64-битные или 128-битные числа. Этот порядковый номер не зависит от назначенного пользователем индивидуального табельного номера сотрудника EmpID. Строковые системы эффективно возвращают данные одной строки или записи за минимально возможное количество дисковых операций. Например, когда программная система пытается получить информацию о конкретном объекте – контактную информацию пользователя или информацию о продукте для системы онлайн-покупок.

Но при выполнении общесистемных операций над всей таблицей строковые системы неэффективны. Например, чтобы найти все записи в таблице примера с окладами от 40 000 до 50 000 СУБД должна полностью просмотреть все записи таблицы в поисках нужных записей. Уже для подобной

таблицы даже с несколькими сотнями строк потребуется несколько обращений к диску для чтения различных блоков. Дело в том, что жесткие диски логически разделены на серию блоков фиксированного размера, в которых обычно хранится несколько строк таблицы. Самыми дорогостоящими операциями при обращении к строковым данным, хранящимся на диске, являются операции поиска. Чтобы прочесть все данные большой таблицы, следует многократно произвести операции чтения блоков.

Какие решения могут помочь минимизировать количество дисковых операций?

1. Для повышения производительности операций чтения большинство СУБД поддерживают использование *индексов*. Индексы сохраняют значения идентификаторов RowID со значениями часто использующихся столбцов. Индекс для столбца с окладом будет выглядеть примерно так:

```
55000: 004;
60000: 001;
80000: 002;
94000: 003;
```

Индексы создаются по одному или нескольким столбцам. Индексы обычно намного меньше, чем хранилища основной таблицы потому что они хранят только отдельные фрагменты данных, а не целые строки. Считывание этого меньшего набора данных сокращает количество дисковых операций. Если индекс используется интенсивно, он может значительно сократить время поиска. Однако поддержание индексов увеличивает нагрузку на систему, особенно когда в базу данных записываются новые данные. В основной таблице должны храниться не только записи, но и любые прикрепленные индексы. Обычно формируется несколько индексов, в зависимости от того, по каким полям требуется делать поиск, что увеличивает объем БД на диске иногда в несколько раз.

2. Существует уже не малое количество баз данных, полностью размещающихся в оперативной памяти. СУБД, управляющие такими данными, не зависят от дисковых операций и имеют равный и быстрый доступ ко всему набору данных, что снижает потребность в индексах. Однако такие СУБД, не смотря на свою относительную простоту и малый размер, управлять только базами данных, которые умещаются в памяти.

3. Для минимизации многократного обращения к диску можно попытаться по-другому организовать хранение данных.

База данных, ориентированная на столбцы, группирует (сериализует) все значения первого столбца, затем значения следующего столбца и т.д. В нашем примере таблицы данные будут храниться следующим образом:

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
```

```
60000:001,80000:002,94000:003,55000:004;
```

Давайте повернём макет так, чтобы каждая строка стала столбцом:

```
10:  001,  Smith:   001,  Joe:  001,  60000:  001,
12:  002,  Jones:   002,  Mary:  002,  80000:  002,
11:  003,  Johnson: 003,  Cathy:  003,  94000:  003,
22:  004;  Jones:   004;  Bob:  55000:
                          004;  004;
```

В этом макете любой из столбцов более точно соответствует структуре индекса в строковой системе. Такое преобразование может вызвать путаницу и привести к ошибочному мнению, что колоночное хранилище «на самом деле является просто» хранилищем строк с индексом для каждого столбца. Однако отображения данных кардинально отличаются. В системе, ориентированной на строки, индексы сопоставляют значения столбцов с идентификаторами строк, тогда как в противоположной системе столбцы сопоставляют идентификаторы строк со значениями столбцов. Это может показаться сложным, но различие можно увидеть в этой общей модификации одного и того же хранилища, где два элемента «Jones», сжимаются в один элемент с двумя идентификаторами строк:

```
…;
Smith: 001,
Jones: 002 004,
Johnson: 003,
…
```

Будет ли система, ориентированная на столбцы, более эффективна в работе – во многом зависит от объёма рабочей нагрузки. Чем больше объем хранимых данных, чем больше аналитической обработки отдельных полей, тем эффективнее колоночная модель организации данных.

Предпосылки к использованию колоночных БД

Насколько эффективно будет работать программная система если необходимо выбрать, например, только три поля из таблицы, в которой их пятьдесят? В силу построчного хранения данных в традиционных СУБД (необходимого для частых операций добавления новых записей в учетных системах) будут прочитаны абсолютно все строки целиком со всеми полями. И не важно, нужны ли для текущей обработки три поля или все поля, в любом случае они все будут прочитаны с диска целиком и полностью, пропущены через контроллер дискового ввода-вывода и переданы процессору, который уже отберет только необходимые для запроса. Каналы дискового ввода-вывода обычно являются основным ограничителем производительности аналитических систем. Как результат, эффективность традиционной СУБД при выполнении данного запроса может снизиться в 10-15 раз из-за неминуемого чтения лишних данных.

Реляционные СУБД, широко применявшиеся в первых учетных программных системах для поддержки хозяйственной деятельности начиная с 1970-х гг., надежно хранили (достаточно скромные по современным меркам) все входящие сведения и, при необходимости, быстро их находили. Используемая реляционная модель чрезвычайно удобна для частых операций добавления новых строк в базу данных – новая запись может быть добавлена целиком всего за один проход головки дискового накопителя. Такие требования обусловили архитектурные особенности реляционных СУБД, оставшиеся до настоящего времени практически неизменными: построчное хранение данных, индексирование записей и журналирование операций. Однако в 1990-х, с распространением аналитических информационных систем и хранилищ данных, применявшихся для управленческого анализа накопленных в учетных системах сведений, стало понятно, что характер нагрузки в этих двух видах систем радикально отличается. Транзакционным приложениям свойственны очень частые мелкие операции добавления и изменения одной или нескольких записей (insert/update).

В аналитических системах наибольшая нагрузка создается сравнительно редкими, но тяжелыми выборками (select) сотен тысяч и миллионов записей, часто с группировками и расчетом итоговых значений (так называемых *агрегатов*). Количество операций записи при этом невысоко, нередко менее 1% общего числа. Причем часто запись идет крупными блоками (bulk load). Для реализации аналитической обработки, как правило, требуется совсем небольшое число полей. В среднем, в аналитическом SQL-запросе пользователя их редко бывает больше 7-8. А операции по извлечению всех хранимых данных целой строкой обычно редки.

Исходя из вышесказанного, реляционные СУБД, ориентированные на оперативную обработку транзакций в реальном времени (Online Transaction Processing, OLTP), больше ориентированы на строки. В то время как столбцово-ориентированные СУБД очень эффективны для систем онлайн-аналитики (Online Analytical Processing, OLAP), позволяя вычислять статистику по столбцам на один-два и более порядков быстрее, чем это делается при использовании традиционных БД.

Другие примеры в пользу колоночной модели данных. В большинстве современных систем онлайн-торговли необходимо извлечь только ограниченный набор атрибутов объектов (только цвет, только размер, только стоимость, только материал, и т.д.) и выполнить на них какую-то несложную статистическую обработку. Сбор только имени и фамилии из множества всех данных о пользователе для построения списка контактов гораздо более распространен, чем чтение всех его данных. Наоборот, для вычисления среднего возраста респондентов совсем не нужны данные о его имени и фамилии и другие атрибуты, хранящиеся в одной таблице реляционной БД. При работе с клиентской базой данных, содержащей двадцать миллионов записей, связанных с информацией о покупке, типичный запрос о покупке в таблице заказов с использованием MySQL MyISAM занимет 30-40 минут. Те же

запросы с использованием MariaDB AX (вариант колоночного хранения MySQL) занимают от 0,1 секунды до 8 секунд.

Семейства столбцов

В столбчатых хранилищах столбцы могут объединяться в определенные группы или *семейства столбцов*. Каждое семейство – это набор логически связанных столбцов, которые обычно извлекаются или управляются как единое целое. Данные, которые совместно используются в иных процессах, хранятся отдельно в других семействах столбцов. Хранилища поддерживающие семейства столбцов, также известны как *базы данных семейств столбцов* (Column family databases) или *расширяемое хранилище записей* (Wide-column stores). В семействе столбцов все данные хранятся построчно, так что столбцы для данной строки хранятся вместе, а не каждый столбец хранится отдельно.

В семейство столбцов можно достаточно просто добавлять новые столбцы, а строки при этом могут быть разреженными, т.е. строки не обязаны иметь значение для каждого столбца. Семейства могут состоять из практически неограниченного количества колонок, которые могут создаваться во время работы программы (динамически) или во время проектирования модели хранения данных.

На рисунке 5.10 представлен пример таблицы с двумя семействами столбцов: Identity и Contact Info. Данные одной сущности имеют одинаковые ключи строк во всех семействах столбцов. Такая структура, в которой строки любого объекта в семействе столбцов могут динамически изменяться, определяет важное преимущество этой категории хранилищ. Семейства столбцов очень хорошо подходят для хранения данных с различными схемами.

CustomerID	Column Family: Identity
001	First name: Ivan Last name: Kuznetsov
002	First name: Vitaliy Last name: Popov Suffix Jr.
003	First name: Elena Last name: Ivanova Title Dr.

CustomerID	Column Family: Contact Info
001	Phone number: 333-0100 Email: ivkuz@examlple.com
002	Email: popovV@cosmo.com
003	Phone number: 355-0020

Рисунок 5.10. Пример данных семейств столбцов

В отличие от хранилища «Ключ –значение» и баз данных документов, большинство столбчатых баз данных упорядочивают хранимые данные с

помощью самих значений ключей, а не хэш-кодов от них. Ключ строки рассматривается как *первичный индекс* и обеспечивает доступ на основе определенного ключа или их диапазона. Некоторые реализации позволяют создавать *вторичные индексы* по определенным столбцам в семействе столбцов. Вторичные индексы позволяют получать данные по значениям столбцов, а не ключам строки.

Сжатие в колоночных БД

Данные столбца имеют единый тип, поэтому имеются возможности для оптимизации хранимых данных и размера самого хранилища. Например, многие популярные современные схемы сжатия, такие как LZW (Lempel-Ziv-Welch, LZW – универсальный алгоритм сжатия данных без потерь) или кодирование длин серий, используют для сжатия сходство смежных данных. Отсутствующие значения и повторяющиеся значения, часто встречающиеся в клинических данных, могут быть представлены небольшими битовыми маркерами. Определенные методы можно использовать и для сжатия строковых данных, хотя их реализация дает менее эффективные результаты.

Улучшению сжатия может способствовать сортировка строк. Например, с помощью индексов на основе битовых карт (bitmap index) сортировка может улучшить сжатие на порядок.

Однако столбцовое сжатие позволяет уменьшить дисковое пространство за счет снижения эффективности поиска. Чем выше сжатие, тем сложнее может стать произвольный доступ, поскольку для чтения могут потребоваться несжатые данные. Поэтому колоночные архитектуры иногда дополняются дополнительными механизмами, направленными на минимизацию доступа к сжатым данным.

Преимущества и ограничения колоночных хранилищ

Сравнение между колоночными и реляционными БД обычно связано с эффективностью доступа к жесткому диску для определенной рабочей нагрузки, поскольку время поиска невероятно велико по сравнению с другими узкими местами в компьютерных системах. Например, типичный жесткий диск с интерфейсом Serial ATA (SATA) имеет среднее время поиска от 16 до 22 миллисекунд, в то время как доступ к оперативной памяти DRAM на процессоре Intel Core i7 занимает в среднем 60 наносекунд, что почти в 400 000 раз быстрее. Очевидно, что при работе с большими данными доступ к диску является основным узким местом. В этом случае выигрывают столбчатые базы данных – они повышают производительность за счет уменьшения объема данных, которые необходимо прочитать с диска: 1) за счет эффективного сжатия одинаковых столбчатых данных; 2) за счет чтения только тех столбцов данных, которые необходимы для ответа на запрос.

На более дешевом и маломощном оборудовании колоночные БД позволяют получить прирост скорости выполнения запросов в 5, 10 и иногда даже в 100 раз, при этом, благодаря компрессии, данные будут занимать на диске в 5-10 раз меньше, чем в случае с традиционными СУБД. Причем объемы данных могут быть достаточно большими – есть примеры по 300-500ТБ и даже случаи с более чем 1ПБ данных.

Колонки – это просто отдельные файлы, добавление и удаление колонок ничего не стоит. Это просто создание и удаление файлов на диске. В случае же строчной БД новая колонка приводит к обновлению данных в каждой строке таблицы.

Однако для записи данных в столбчатую базу данных необходимо проделать существенную работу. При реализации операции записи (INSERT) необходимо вносить данные в каждый столбец, сами столбцы сжимать по мере необходимости. Другими словами, стоимость одиночных вставок очень высока. При этом следует учитывать, как именно производится запись: часто, но построчно, или редко, но большим массивом (bulk load). Но в общем случае — в разы, а может быть даже в десятки раз медленнее, чем запись тех же объемов в традиционных БД.

Не все колоночные СУБД обеспечивают должную аналитическую производительность. Некоторые из них (HBase, Cassandra) просто хранят отдельные столбцы, сгруппированные вместе, чтобы поддерживать свою распределенную таблицу (большую таблицу) и относительно быструю запись.

Колоночные СУБД

Колоночные хранилища были реализованы наряду с появлением первых дней разработки СУБД. Первой коммерческой системой хранения, ориентированной на столбцы, с упором на хранение и поиск информации в биологии, была система TAXIR, созданная в 1969 году. Клинические данные из историй болезни пациентов с гораздо большим количеством атрибутов, чем можно было проанализировать, были обработаны в 1975 году, а затем спустя некоторое время появилась система с колоночной БД TODS. Статистическое управление Канады внедрило систему RAPID в 1976 г. для обработки и извлечения данных переписи населения и жилищного фонда Канады, а также для ряда других статистических приложений. Она с успехом использовалась до конца 90-х гг. RAPID был передан другим статистическим организациям во всем мире и широко использовался в 80-х годах.

Современные колоночные СУБД преодолевают узкие места первых систем за счёт загрузки данных большими партиями, что оптимизирует процесс записи. Такой вариант возможен когда много мелких операций записи производится в традиционную БД, а потом с нужной периодичностью перегружается в колоночную. Версии некоторых современных колоночных СУБД научились хорошо работать и с потоком мелких операций записи в реальном времени.

Хранящиеся на диске данные практически всегда сжимаются для преодоления проблем неэффективного использования дискового пространства. В дополнение к сжатию большинство современных колоночных хранилищ хранят данные столбцов в блоках с несколькими предварительно вычисленными статистическими данными. Это позволяет выполнять запросы намного быстрее, и производить вычисления, которые требуют обработки гораздо меньшего количества данных.

Большинство, хотя и не все, современные колоночные хранилища предназначены для работы на распределенном кластере серверов.

Популярные колоночные СУБД

D рейтинге DB_Engines базовые колоночные системы не выделяются, а относятся к реляционным, многие коммерческие реляционные СУБД включают в себя и этот вид модели как отдельную особенность.

Перечень тринадцати популярных специализированных расширяемых колоночных хранилищ (Wide column store) на декабрь 2021 г. представлен в таблице 5.4. Некоторые хранилища поддерживают и другие NoSQL-модели.

Таблица 5.4. Популярность колоночных СУБД

№	СУБД	Тип	Другие поддерживаемые типы	Рейтинг
1.	Cassandra	Wide column store		119.20
2.	HBase	Wide column store		45.54
3.	Microsoft Azure Cosmos DB	Wide column store Multi-model	Document store Graph DBMS Key-value store Spatial DBMS	39.71
4.	Datastax Enterprise	Wide column store Multi-model	Document store Graph DBMS Spatial DBMS Search engine	9.28
5.	Microsoft Azure Table Storage	Wide column store		5.28
6.	ScyllaDB	Wide column store Multi-model	Key-value store	3.93
7.	Accumulo	Wide column store		3.91
8.	Google Cloud Bigtable	Wide column store		3.63
9.	HPE Ezmeral Data Fabric	Wide column Multi-model	Document store	0.89
10.	Elassandra	Wide column store Multi-model	Search engine	0.62
11.	Amazon Keyspaces	Wide column store		0.54
12.	Alibaba Cloud Table Store	Wide column store		0.40

№	СУБД	Тип	Другие поддерживаемые типы	Рейтинг
13.	SWC-DB	Wide column store Multi-model	Time Series DBMS	0.06

Cassandra. Распределённая СУБД, рассчитанная на создание высоко масштабируемых и надёжных хранилищ огромных массивов данных, хорошо подходящая для гибридных и многооблачных сред. по аналогии Изначально проект был разработан в недрах Facebook, но в 2009 году передан в Apache Software Foundation, эта организация продолжает развитие проекта. Лицензия Apache, Open Source-продукт. Промышленные решения на базе Cassandra развёрнуты для обеспечения сервисов таких компаний, как Cisco, IBM, Huawei, Netflix, Apple, Instagram, GitHub, Twitter Spotify и др. К 2011 году крупнейший кластер серверов, обслуживающий единую базу данных под управлением Cassandra, насчитывал более 400 машин и содержал данные размером более 300 ТБ.

Написана на языке Java, реализует распределённую хэш-систему, сходную с СУБД DynamoDB, что обеспечивает практически линейную масштабируемость при увеличении объёма данных. Использует модель хранения данных на базе семейства столбцов. Поддерживаемые языки программирования: C#, C++, Clojure, Erlang, Go, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby, Scala.

Типичные области применения: интернет вещей (IoT), приложения для обнаружения мошенничества, механизмы рекомендаций, каталоги продуктов и списки воспроизведения, приложения для обмена сообщениями.

HBase. Распределенная СУБД с открытым исходным кодом, написанная на языке Java по аналогии BigTable от Google. Изначально эта СУБД класса NoSQL создавалась компанией Powerset в 2007 году для обработки больших объёмов данных в рамках поисковой системы на естественном языке. Проектом Apache Software Foundation HBase стала в 2010 году. Лицензия Apache, Open Source-продукт.

Используется для управления данными в ряде крупных проектов, в частности, Facebook в период 2010-2018 годов использовал HBase для платформы сообщений; к числу постоянных пользователей также относятся Adobe, StumbleUpon, Twitter. Компания Yahoo! эксплуатирует HBase-кластер из 3000. узлов.

Относится к категории «семейство столбцов» (wide-column store) и представляет собой колоночно-ориентированное, мультиверсионное хранилище типа «ключ-значение» (key-value). Она работает поверх распределенной файловой системы HDFS и обеспечивает возможности BigTable для Hadoop, реализуя отказоустойчивый способ хранения больших объёмов разреженных данных. Поддерживаемые языки программирования: те же, что и у Cassandra.

Модель данных HBase реализуясь по типу ключ-значение:

$<table, RowKey, Column\ Family, Column, timestamp> \rightarrow Value$

Данные организованы в таблицы (table), проиндексированные первичным ключом (RowKey); для каждого первичного ключа может храниться неограниченный набор колонок; колонки организованны в группы колонок (Column Family). Обычно в одну группу объединяют колонки с одинаковым шаблоном использования и хранения (Column). Список и названия групп колонок фиксирован и имеет четкую схему. На уровне группы колонок задаются такие параметры как time to live (TTL) и максимальное количество хранимых версий.Для каждого атрибута может храниться несколько различных версий. Разные версии имеют разный штамп времени (timestamp).

Записи физически хранятся в порядке, отсортированном по первичному ключу. При этом информация из разных колонок хранится отдельно, благодаря чему можно считывать данные только из нужного семейства колонок, таким образом, ускоряя операцию чтения. Атрибуты, принадлежащие одной группе колонок и соответствующие одному ключу, физически хранятся как отсортированный список. Любой атрибут может отсутствовать или присутствовать для каждого ключа. Отсутствие атрибута не влечет никаких накладных расходов на хранение пустых значений.

LucidDB. Open Source СУБД, основанное на колоночной технологии. Позиционируется как замена MySQL для аналитических задач. Ориентирована на OLAP серверы и системы бизнес-логики. Поддерживает индексирование Bitmap, агрегирование/добавление хэша и многовариантности на уровне страниц. В отличие от других баз данных, каждый компонент LucidDB был спроектирован с целью увеличения гибкости и производительности. Для работы LucidDB не требуется администратор базы данных.

Обрабатывает функции ETL (извлечение, преобразование и загрузка данных) с использованием расширений ANSI SQL, используя «оболочки» для ряда источников данных (базы данных, текстовые файлы, веб-службы и т. д.), обращаясь к ним так, как если бы они все были реляционной БД.

LucidDB долгое время была первой чистой СУБД колоночного хранилища с открытым исходным кодом. Однако при отсутствии коммерческих спонсоров и постоянной активности сообщества прекращено развитие и официальная поддержка проекта.

Поддержка колоночных стратегий в реляционных СУБД

Классические реляционные СУБД могут использовать колоночные стратегии путем смешивания строковых таблиц и столбцов. Несмотря на сложность таких СУБД, этот подход доказал свою ценность с 2010 года по настоящее время. Например, в 2014 г. корпорация Citus Ddata представила

расширение СУБД PostgreSQL, с функцией хранения гибридных столбцов. Эта функция позволяет выбирать некоторые столбцы из таблиц БД, а остальные хранить традиционным способом. Компания McObject в 2012 г. добавила поддержку столбчатого хранилища в своей СУБД eXtremeDB Financial Edition.

Microsoft Azure Cosmos DB – мультимодельная СУБД, запущенная в мае 2017 года, поддерживает создание *аналитического хранилища столбцов*, которое создается на основе операционных данных. Это хранилище столбцов сохраняется отдельно от транзакционного хранилища строк и поэтому аналитические запросы не влияют на производительность транзакционных рабочих нагрузок. Операции вставки, обновления и удаления для операционных данных автоматически синхронизируются с аналитическим хранилищем в реальном времени. Задержка автоматической синхронизации обычно составляет менее 2 минут. Для синхронизации не требуется канал изменений или ETL.

Благодаря хранению данных в виде столбцов аналитическое хранилище позволяет группировать значения для каждого поля, чтобы сериализовать их вместе. Такой формат сокращает количество операций ввода-вывода, необходимых для сканирования или вычислений статистики по конкретным полям. Он значительно сокращает время отклика запросов для больших наборов данных.

Пример формирования строчной и колоночной моделей данных для рабочей таблицы в среде Azure Cosmos DB представлен на рисунке 5.11. Хранилище строк сохраняет данные приведенной таблицы в сериализованном строковом формате на диске. Такой формат позволяет быстрее выполнять транзакционные операции чтения, записи и оперативные запросы, например, «Вернуть сведения о product1». По мере роста набора данных и необходимости выполнять сложные аналитические запросы к отдельным атрибутам, подобные запросы могут оказаться дорогостоящими. Например, необходимо получить тенденции продаж для продукта в категории «Оборудование» по разным подразделениям и месяцам. Для этого необходимо выполнить сложный запрос. Длительные проверки большого набора данных могут оказаться затратными в плане пропускной способности и могут повлиять на производительность транзакционных рабочих нагрузок, обеспечивающих работу приложений и служб в режиме реального времени.

Аналитическое хранилище, которое является хранилищем столбцов, лучше подходит для таких запросов, так как оно сериализует логически смежные поля данных вместе и сокращает число операций ввода-вывода на диск.

ID	Product Name	Product Code	Product Category
1	Product1	123	Books
2	Product2	250	Equipment
3	Product3	380	Equipment

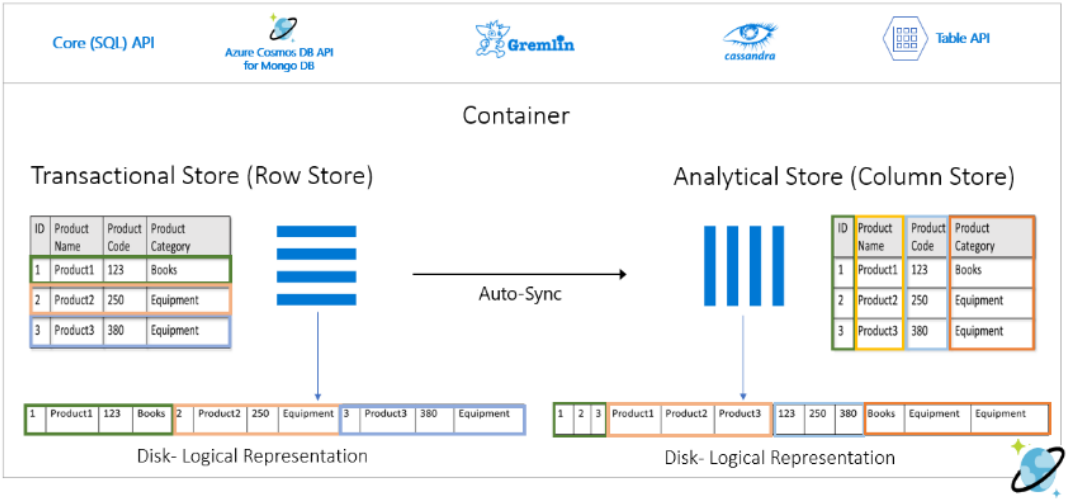

Рисунок 5.11. Преобразование транзакционной модели в колоночную модель в СУБД Azure Cosmos DB

ЗАКЛЮЧЕНИЕ

Мы рассмотрели достаточно много материала, чтобы сделать краткий обзор трех разных NoSQL-моделей. Все эти модели можно отнести к разряду *агрегатно-ориентированных*. Агрегат – это термин, пришедший из предметно-ориентированного проектирования. В предметно-ориентированном проектировании агрегатом называют коллекцию связанных объектов. Эта коллекция интерпретируется как единое целое.

Применительно к хранению данных агрегат представляет собой единицу для манипулирования данными и управления их согласованностью. Обычно агрегаты модифицируются с помощью атомарных операций, а взаимодействие с хранилищем данных осуществляется посредством пересылки агрегатов. Это определение довольно точно описывает принципы работы баз данных типа «Ключ – значение», документо-ориентированных и колоночных. Агрегаты облегчают работу баз данных на кластерах, поскольку агрегат представляет собой естественную единицу репликации и фрагментации. Кроме того, агрегаты упрощают разработку прикладных программ, которые часто манипулируют данными с помощью агрегированных структур. Для рассмотренных СУБД характерно понятие агрегата, индексированного ключом, который можно использовать для поиска. Агрегат очень важен для работы на кластерах, поскольку база данных в этом случае будет гарантировать, что все данные одного агрегата хранятся на одном узле. Кроме того, агрегат является

атомарной единицей модификации, обеспечивая полезный, хотя и ограниченный объем управления транзакциями.

Агрегаты бывают разными. Модели данных типа «Ключ-значение» интерпретируют агрегаты как «черный ящик», т.е. искать можно только целые агрегаты, невозможно подать запрос на извлечение части агрегата. В документной модели агрегат является прозрачным для базы данных. Это позволяет посылать запросы к фрагментам агрегата и осуществлять частичное извлечение данных. Однако, поскольку документ не имеет схемы, база данных не может сильно влиять на структуру документа, чтобы оптимизировать хранение и извлечение частей агрегата. Модели типа «семейство столбцов» разделяют агрегат на группы столбцов, интерпретируя их как единицы данных в агрегате-строке. Это накладывает на агрегат структурные ограничения, но позволяет базе данных использовать эту структуру для улучшения доступа.

Большинство хранилищ данных выполняют обработку запросов и данных на стороне сервера. Иногда функции обработки встроены в подсистему хранилища данных. В других случаях функции обработки данных вынесены в отдельный модуль или несколько модулей для обработки и анализа. Хранилища данных поддерживают разные интерфейсы программного доступа и управления.

Хранилища данных, включенные в одну категорию, не обязательно предоставляют одинаковый набор возможностей. А особенностью многих современных СУБД является поддержка сразу несколько моделей хранения данных. И этот процесс неуклонно развивается.