Оглавление

Введение

Многомодельная база данных — это база данных, предназначенная для поддержки нескольких моделей данных в одной системе хранения данных. Это означает, что такая система может хранить, индексировать и запрашивать данные в нескольких моделях. Этот тип базы данных обеспечивает единый интерфейс для обеспечения согласованности, безопасности и доступа к данным, а также устраняет необходимость сложных преобразований и миграций между различными базами данных [1].

В настоящее время существует четыре основные модели баз данных: ключ-значение, семейство столбцов, документальные и графовые [2].

Базы данных ключ-значения имеют хранилище ключ-значение. Они позволяют разработчику приложения хранить данные без схемы. Эти данные обычно состоят из строки, которая представляет ключ, и фактических данных, которые считаются значением в отношениях «ключ-значение». Сами данные обычно представляют собой своего рода примитив языка программирования (строка, целое число, массив) или объект, который создаётся привязками языков программирования к хранилищу ключ-значение. Это заменяет необходимость в фиксированной модели данных и делает строгими требования к правильному форматированию данных без данных [3].

Графовая модель базы данных — это модель, в которой структуры данных для схемы и/или экземпляров моделируются как направленный, возможно, помеченный граф или обобщение структуры данных графа, где манипулирование данными выражается с помощью графо-ориентированных операций и конструкторов типов, а соответствующие ограничения целостности могут быть определены в структуре графа [4].

База данных семейства столбцов - это база данных NoSQL, которая хранит данные с использованием столбцового подхода, в отличие от реляционных, которые упорядочивают данные по строкам. Данные, хранящиеся в базе данных семейства столбцов, выбираются вертикально, что делает частичное чтение более эффективным, поскольку загружается только набор атрибутов строки [5].

Вместо хранения данных в фиксированных строках и столбцах базы данных документов используют гибкие документы. Документ – это запись в базе данных документов. Документ обычно хранит информацию об одном объекте и любых связанных с ним метаданных. Документы хранят данные в парах поле-значение. Значения могут быть различных типов и структур, включая строки, числа, даты, массивы или объекты [6].

Далее представлены некоторые из наиболее популярных мультимодельных баз данных:

1. ArangoDB — это бесплатный менеджер баз данных с открытым исходным кодом, который поддерживает модели баз данных ключ-значение, документ и граф [7].
2. OrientDB поддерживает следующие модели баз данных: граф, документа, ключ-значение [8].
3. Azure Cosmos DB поддерживает следующие модели баз данных: документ, ключ-значение, граф [9].

Многомодельная обработка и оптимизация запросов. Несмотря на то, что многомодельные БД способны хранить данные в различных форматах (моделях), они не обеспечивают язык межмодельной обработки данных, межмодельную компиляцию или соответствующую оптимизацию многомодельных запросов. Напротив, многомодельная база данных пытается решить эту проблему путем разработки унифицированного языка

запросов, который будет охватывать все поддерживаемые модели данных. Однако существующие языки запросов еще незрелы, и разработка полноценного языка запросов для многомодельных данных все еще остается открытой задачей. Тесно связанной проблемой является предложение подхода к определению оптимального плана запроса для эффективной оценки заданный межмодельный запрос.

Проектирование и оптимизация многомодельной схемы. Хороший дизайн схемы базы данных является важной частью, влияющей на многие аспекты, такие как эффективность обработки запросов, расширяемость приложения и т.д. В отличие от реляционных баз данных, базы данных NoSQL обычно используют значительно денормализованную физическую схему, которая требует дополнительного места. Следовательно, в многомодельных системах мы сталкиваемся с противоречивыми требованиями к различным моделям, и, таким образом, требуется новое решение для проектирования многомодельных схем, чтобы сбалансировать и найти компромисс между разнообразными требованиями к многомодельным данным. Даже вопрос существования схемы существенно различается: традиционные реляционные базы данных основаны на существовании заранее определенной схемы, тогда как базы данных NoSQL основаны на предположении об отсутствии схемы.

Мультимодельная эволюция. В целом, эффективное управление развитием схемы данных и распространением изменений в соответствующих частях системы базы данных, таких как экземпляры данных, запросы, индексы или даже стратегии хранения, является сложной задачей. В некоторых небольших приложениях компания может положиться на квалифицированный администратор базы данных для управления развитием данных и распространения изменений на другие затронутые части вручную. Но в большинстве случаев это сложная и

подверженная ошибкам работа. В контексте многомодельных баз данных эта задача более тонкая и трудная.

Многомодельная расширяемость. Последней открытой проблемой является проблема расширяемости модели, которую можно рассматривать в нескольких аспектах. Во-первых, мы можем рассмотреть возможность расширения внутри модели, что означает расширение одной из моделей новыми конструкциями. Во-вторых, мы можем рассмотреть межмодельную расширяемость, которая добавляет новые конструкции, выражающие отношения между моделями. В-третьих, мы можем обеспечить дополнительную расширяемость модели, которая включает добавление совершенно новой модели вместе с соответствующими данными и запросом [10].

Методы построения мульмодельных БД

Можно выделить 4 метода построения мультимодельных БД: Polyglot persistence, мультимодельные СУБД на основе реляционной модели, мультимодельные СУБД на основе документной модели, СУБД «без основной модели» [11].

1) Polyglot persistence

Одним из самых известных методов построения БД является Polyglot persistence. Этот метод подразумевает выбор столько баз данных, сколько необходимо, чтобы все требования были удовлетворены. Данный метод может быть оптимальным решением, когда необходимо обеспечить обратную совместимость с устаревшим приложением. Новая система баз данных может работать параллельно с устаревшей системой баз данных; хотя устаревшее приложение по-прежнему остается полностью функциональным, новые требования могут быть учтены при использовании новой системы баз данных. Очевидно, следует избегать перекладывания

бремени всех этих задач обработки запросов и синхронизации базы данных на уровень приложения, то есть, в конечном итоге, на программистов, которые поддерживают приложения для обработки данных. Вместо этого обычно лучше ввести уровень интеграции. Затем уровень интеграции занимается обработкой запросов: разбивает запросы на несколько подзапросов, перенаправляет запросы в соответствующие базы данных и повторно объединяет результаты, полученные из баз данных, к которым осуществляется доступ. Более того, уровень интеграции должен обеспечивать согласованность между базами данных: он должен синхронизировать данные в различных базах данных путем распространения дополнений, модификаций или удалений между ними. Однако многоязычие сопряжено с серьезными недостатками:

- Отсутствие одинокого способа доступа к данных. Нет уникального интерфейса запроса или языка запросов, и, следовательно, доступ к системам баз данных не унифицирован и требует знания всех необходимых методов доступа к базе данных.
- Согласованность. Согласованность между базами данных является серьезной проблемой, поскольку необходимо обеспечить ссылочную целостность во всех базах данных (например, если запись в одной базе данных ссылается на запись в другой базе данных), а в случае дублирования данных (и, следовательно, в разных представлениях в нескольких базах данных одновременно) дубликаты должны быть обновлены или удалены практически одновременно. Базовые системы баз данных разрабатываются независимо. Более новые версии баз данных могут быть несовместимы с уровнем интеграции, и администратору приходится отслеживать частые обновления.

- Логическая избыточность. Логической избыточности можно избежать только при разработке базы данных, которая строго распределяет непересекающиеся подмножества данных по разным базам данных. Это может противоречить некоторым требованиям пользователей к доступу.
- Безопасность. Уровень интеграции должен обеспечивать контроль доступа, и все подключенные базы данных должны быть настроены так, чтобы разрешать только ограниченный доступ [12].

2) Мультимодельные СУБД на основе реляционной модели

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

3) Мультимодельные СУБД на основе документной модели

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

4) СУБД «без основной модели»

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

Обзор мультимодельных БД

MongoDB

MongoDB – это не реляционная, а документно-ориентированная система управления базами данных. Документно-ориентированная СУБД заменяет концепцию «строки» более гибкой моделью, «документом». Позволяя использовать вложенные документы и массивы, документно-

ориентированный подход дает возможность представлять сложные иерархические отношения с помощью одной записи. Также нет предопределенных схем: ключи и значения документа не имеют фиксированных типов или размеров. Когда нет фиксированной схемы, добавлять или удалять поля по мере необходимости становится проще. MongoDB – СУБД общего назначения, поэтому помимо создания, чтения, обновления и удаления данных она предоставляет большинство тех функций, которые можно ожидать от системы управления базами данных. Специальные типы коллекций и индексов MongoDB поддерживает коллекции данных TTL (time-to-live), срок действия которых должен истечь в определенное время, такие как сеансы и коллекции фиксированного размера, для хранения недавно полученных данных, например, журналов. MongoDB также поддерживает частичные индексы, ограниченные только теми документами, которые соответствуют фильтру критериев, чтобы повысить эффективность и уменьшить необходимый объем дискового пространства [13].

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

Для простого добавления элемента в коллекцию используется метод insert:

db.collection('name_ collection ').insert({ “elem_name” : “elem_value” });

Для добавления множества элементов необходимо через запятую указать нужные:

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

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

db.collection('name_ collection ').remove();

Удалить объект по какому-то параметру необходимо следующим образом:

db.collection('name_ collection ').remove({ “elem_name” : “elem_value” });

Для обновления коллекции используется метод update:

db.collection('name_ collection ').update({ “elem_name” : ” elem_value” });

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

db.collection('name_ collection ').update({ $set: { “elem_name”: ” new_elem_value” }});

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

db.collection('name_ collection ').update({ $unset: { “elem_name”: ” elem_value” }});

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

db.collection('name_ collection ').update({$set: “elem_name”: ” new_elem_value”}, true);

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

db.collection('name_ collection ').find({ “elem_name” : ” elem_value”});

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

Так же имеется возможность использовать для поиска регулярные выражения [14].

Использует протокол — MongoDB Wire. Это простой протокол в стиле запроса-ответа, основанный на сокетах. Клиенты взаимодействуют с сервером базы данных через обычный сокет TCP/IP [15].

Одной из больших проблем выделяют отсутствие схемы данных, что затрудняет расширение и валидацию данных. Так же отмечается уменьшение производительности при большом количестве данных. Выделяют сложность поиска из-за языка запросов [16].

ArangoDB

ArangoDB поддерживает три модели данных: графовую, документную и модель «ключ-значение». Работа с базой данных осуществляется при помощи SQL-подобного языка запросов AQL (ArangoDB Query Language). Язык является декларативным и позволяет свободно комбинировать все поддерживаемые модели данных в одном запросе [17].

Для ArangoDB разработан свой собственный язык запросов - AQL.

Для добавления значений используется следующая команда:

INSERT {key: value} INTO collection [RETURN NEW]

RETURN NEW позволяет вернуть добавленный объект

Для чтения объектов используется:

RETURN DOCUMENT(_id) – вернёт объект по _id

RETURN DOCUMENT(collection_name, [_key, ...]) – вернёт список объектов из коллекции по ключам

Для обновления объектов используется:

UPDATE {_key: "value"} WITH {new_value: 1234} IN collection

Для удаления используется:

REMOVE {_key: "value"} IN collection

Для получения данных используется RETURN, но с особенностями, дял понимание которых далее приведён пример, где из коллекции users выбираются имена пользователей, которые являются активными:

FOR u IN users

FILTER u.active == true

RETURN u.name

В этом примере запроса термины FOR, FILTER и RETURN инициируют операцию более высокого уровня в соответствии с их переводом [18].

ArangoDB предоставляет свой API через HTTP. При необходимости связь может быть зашифрована с помощью SSL [19].

Среди проблем отмечают маленькую популярность, что усложняет решение возникающих вопросов или проблем. Так же иногда отмечают недостаток функционала и, в редких случаях, неполноты документации [20].

OrientDB

OrientDB - это система управления базами данных NoSQL с открытым исходным кодом, написанная на Java. Это мультимодельная база данных, поддерживающая графическую, документальную, ключ/значение и объектную модели. Он поддерживает режимы без схемы, с полной схемой и со смешанной схемой. Он имеет систему профилирования безопасности, основанную на пользователях и ролях, и поддерживает запросы с помощью Gremlin наряду с расширенным SQL для обхода графа [21].

В OrientDB таблицы представляются в виде классов, строки - записи в таблице, а свойства класса - поля. В основном OrientDB использует документальную модель, но также можно использовать графовую модель.

Для создания классов используется следующая команда:

CREATE CLASS <class_name>

Для добавления полей в класс, можно использовать следующую команду:

INSERT INTO <class_name> (<field_name>) VALUES (field_value)

Для удаления записей используется команда DELETE:

DELETE FROM <target-name>

Для изменения значения используется UPDATE:

UPDATE Profile SET nick = 'Luca'

Для выбора результатов используется SELECT [22]:

SELECT name, age FROM Account

Использует протокол Binary Protocol.

После установления соединения клиент может подключиться к серверу или запросить открытие базы данных Database Open. В настоящее время поддерживаются только необработанные сокеты TCP/IP. После подключения и открытия базы данных все запросы клиента отправляются на сервер до тех пор, пока клиент не закроет сокет. Когда сокет закрыт, экземпляр сервера OrientDB освобождает ресурсы, используемые для подключения.

Первая операция, следующая за подключением на уровне сокета, должна быть одной из:

1. Подключение к серверу для работы с экземпляром сервера OrientDB

2. Открытие существующей базы данных

В обоих случаях клиенту отправляется обратно идентификатор сеанса (Session ID). Сервер присваивает клиенту уникальный идентификатор сеанса. Это значение должно использоваться для всех дальнейших операций с сервером [23].

Среди проблем выделяют следующие: плохая документация, плохая обратная совместимость. Так же отмечают огромное количество дефектов, что разработчики обещают исправить в новых версиях, так как сделают основной целью сделать продукт более стабильным [24].

MarkLogic

MarkLogic - это документально-ориентированная база данных, разработанная MarkLogic. Это многомодельная база данных NoSQL, которая развилась из базы данных XML для хранения документов JSON и троек RDF [25].

Фактически любая БД в MarkLogic — это виртуальная файловая система, с каталогами, разграничением доступа, временными метками и т.д. Каждый файл — документ XML, проиндексированный сервером; поиск осуществляется внутри любого XML с учётом его разметки. В качестве языка для запросов используется XQuery.

Для добавления XML-тег в указанное местоположение XML-пути используется функция modify. Для этой команды требуются два параметра: новый XML-тег со значением, которое будет вставлено; и XML-путь, по которому будет опубликован новый XML-тег:

xml.modify(insert "new_tag" into "xml_path(path)")

В функции modify() параметр Delete позволяет удалять XML-теги в столбце XML или переменной XML. Для удаления целевых XML-тегов запросу требуется XML-путь.

xml_data.modify('delete (/user/age)')

Для получения значений можно использовать fn:doc, которая извлекает документ, используя URI, указанный в виде xs:string, и возвращает соответствующий узел документа:

fn:doc($uri)

Если $uri - пустая последовательность, результатом будет пустая последовательность.

Если $uri является относительной ссылкой на URI, она определяется относительно значения статического базового свойства URI из статического контекста. Результирующий абсолютный URI преобразуется в xs:string.

Или можно использовать fn:collection, которая возвращает последовательность элементов, идентифицируемых URI коллекции; или коллекцию по умолчанию, если URI не указан.

fn:collection($arg)

Если $arg не указан, функция возвращает последовательность элементов в коллекции по умолчанию в динамическом контексте.

Если значение $arg является относительным xs:anyURI, оно сопоставляется со значением свойства base-URI из статического контекста.

Если $arg - пустая последовательность, функция ведет себя так, как если бы она была вызвана без аргумента [26].

MarkLogic использует протокол запроса данных XML (XDQP)для внутренней связи между узлами в кластере.

XDQP - это прикладной протокол, который работает на уровне TCP, используя порты 7999 и 7998.

Соединения XDQP создаются при запуске MarkLogic Server и, за исключением каких-либо исключительных условий, остаются постоянными до завершения работы сервера. Во время работы кластера в стационарном режиме соединения XDQP не открываются и не закрываются.

Во время запуска для каждого узла в кластере сервер MarkLogic создаст 3 подключения к другим узлам на их порту 7999. Симметрично другой узел также создаст 3 подключения к порту 7999 первого узла.

Сервер MarkLogic отправляет сообщение heartbeat на каждый узел кластера каждую секунду. Сообщение heartbeat синхронизирует все серверы с одинаковыми часами, поддерживает согласованное состояние

"quorum", распространяет изменения конфигурации и может содержать данные запроса.

Сообщения о “сердцебиении” передаются циклически по всем 3 каналам подключения. Если сообщения о “сердцебиении” прерываются на длительный период времени, узел может отключиться от кластера (тайм-аут узла).

Новые подключения отклоняются, если обнаруживается превышение допустимого предела синхронизации (тайм-аут хоста). Другой хост объявляется недоступным, если проблема сохраняется при новом подключении [27].

Среди проблем выделяют очень высокую стоимость лицензирования и необходимость в большом объем пространства, необходимого для хранения данных. Так же отмечают большой порог входа для людей, которые привыкли к другим БД [28].

Azure Cosmos DB

Azure Cosmos DB - это глобально распределенная служба многомодельных баз данных, предлагаемая Microsoft. Она предназначена для обеспечения высокой доступности, масштабируемости и доступа к данным с низкой задержкой для критически важных приложений [29].

Azure Cosmos DB поддерживает несколько api, которые используются разные языки запросов.

Для Azure Cosmos DB for NoSQL используется язык SQL.

Для Azure Cosmos DB for PostgreSQL так же используется SQL.

Для Azure Cosmos DB for MongoDB используется тот же язык, что и в MongoDB. Подробный его синтаксис был описан ранее [30].

Протокол подключения можно выбрать TCP или HTTPS [31].

Среди проблем отмечается высокая цена использования из-за чего сложнее планировать проект. Так же возникают неудобства из-за ограничений аккаунта, который необходим для использования [32].

Выводы

Было произведено изучение мультимодельных данных, их типы построения. Так же был произведён анализ некоторых мультимодельных баз данных, который включал в себя изучение синтаксиса, определение протокола, через который работает СУБД, и выявлены проблемы, возникающие при работе с каждой из рассмотренных СУБД.

Литература

1. [Электронный ресурс] – 2023 г. – Режим доступа: https://en.wikipedia.org/wiki/Multi-model_database, свободный.
2. The Multi-model Databases – A Review / Ewa Płuciennik & Kamil Zgorzałek; — Springer, Cham: Communications in Computer and Information Science, vol. 716, 2017. — 141–152 с.
3. Key-Value stores:a practical overview / Marc Seeger; — Stuttgart, Germany, 2009.
4. Survey of graph database models / Renzo Angles, Claudio Gutierrez; — ACM Computing Surveys, vol. 40, 2008. — 1–39 с.
5. An Approach for Implementing Online Analytical Processing Systems under ColumnFamily Databases / Abdelhak Khalil and Mustapha Belaissaoui; — IAENG International Journal of Applied Mathematics, vol. 53, 2023. — 31–39 с.
6. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.mongodb.com/document-databases, свободный.

7. [Электронный ресурс] – 2023 г. – Режим доступа: https://docs.arangodb.com/3.11/concepts/data-models/, свободный.
8. [Электронный ресурс] – 2023 г. – Режим доступа: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html, свободный.
9. [Электронный ресурс] – 2023 г. – Режим доступа: https://learn.microsoft.com/en-us/azure/cosmos-db/introduction, свободный.
10. Multi-model Databases : A New Journey to Handle the Variety of Data / Lu , J & Holubová; — ACM Computing Surveys , vol. 52 , no. 3 , 2019. — 55 с.
11. [Электронный ресурс] – 2023 г. – Режим доступа: https://habr.com/ru/articles/462493/
12. Polyglot database architectures = polyglot challenges / Lena Wiese; — Gottingen, Germany: CEUR Workshop Proceedings, vol. 1458, 2015. — 422-426 с.
13. MongoDB: The Definitive Guide / Shannon Bradshaw, Eoin Brazil and Kristina Chodorow. — Boston: O'Reilly Media, Inc., 2019. — 511 с.
14. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.mongodb.com/docs/manual/tutorial/query-documents/, свободный.
15. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.mongodb.com/docs/manual/reference/mongodb-wire-protocol/, свободный.
16. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.g2.com/products/mongodb/reviews, свободный.

17. [Электронный ресурс] – 2023 г. – Режим доступа: https://ru.wikipedia.org/wiki/ArangoDB, свободный.
18. [Электронный ресурс] – 2023 г. – Режим доступа: https://docs.arangodb.com/3.11/aql/fundamentals/syntax/, свободный.
19. [Электронный ресурс] – 2023 г. – Режим доступа: https://docs.arangodb.com/3.11/develop/http-api/general-request-handling/, свободный.
20. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.g2.com/products/arangodb/reviews, свободный.
21. [Электронный ресурс] – 2020 г. – Режим доступа: https://en.wikipedia.org/wiki/OrientDB, свободный.
22. [Электронный ресурс] – 2023 г. – Режим доступа: https://orientdb.org/docs/3.2.x/, свободный.
23. [Электронный ресурс] – 2023 г. – Режим доступа: https://orientdb.com/docs/last/internals/Network-Binary-Protocol.html, свободный.
24. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.g2.com/products/orientdb/reviews, свободный.
25. [Электронный ресурс] – 2023 г. – Режим доступа: https://en.wikipedia.org/wiki/MarkLogic_Server, свободный.
26. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.w3.org/TR/xpath-functions-31/, свободный.
27. [Электронный ресурс] – 2023 г. – Режим доступа: https://help.marklogic.com/knowledgebase/article/View/xml-data-query-protocol-xdqp, свободный.
28. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.g2.com/products/marklogic/reviews, свободный.

29. [Электронный ресурс] – 2023 г. – Режим доступа: https://en.wikipedia.org/wiki/Cosmos_DB, свободный.
30. [Электронный ресурс] – 2023 г. – Режим доступа: https://learn.microsoft.com/en-us/azure/cosmos-db/choose-api, свободный.
31. [Электронный ресурс] – 2023 г. – Режим доступа: https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/sdk-connection-modes, свободный.
32. [Электронный ресурс] – 2023 г. – Режим доступа: https://www.g2.com/products/azure-cosmos-db/reviews, свободный.