Содержание

Введение

Современные информационные системы часто используют разнородные системы управления базами данных (СУБД), что обусловлено различными требованиями к хранению, обработке и масштабируемости данных. Однако взаимодействие между различными СУБД остается сложной задачей из-за различий в моделях данных, языках запросов и архитектурах.

Проблема интеграции гетерогенных баз данных существует давно и требует эффективных решений для обеспечения прозрачного доступа к данным, хранящимся в разных системах. В данной работе рассматриваются основные группы методов взаимодействия между СУБД.

Группы методов взаимодействия между различными СУБД

Проблема взаимодействия между различными СУБД и БД существует уже давно [1]. Далее представлены некоторые способы её решения, поделённые на группы.

Узконаправленные системы

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

DiscoveryLink

Общая архитектура DiscoveryLink является общей для многих гетерогенных систем баз данных, включая TSIMMIS, DISCO, Pegasus, DIOM, HERMES и Garlic. Приложения подключаются к серверу DiscoveryLink с помощью любого из множества стандартных клиентских интерфейсов базы данных, таких как OpenDatabase Connectivity (ODBC) или Java DatabaseConnectivity (JDBC**), и отправляют запросы в DiscoveryLink в стандартном SQL. Информация, необходимая для ответа на запрос, поступает из одного или нескольких источников данных, которые были

идентифицированы в DiscoveryLink через процесс, называемый регистрацией. Источники данных, представляющие интерес для наук о жизни, варьируются от простых файлов данных до сложных доменно-специфичных систем, которые не только хранят данные, но и включают специализированные алгоритмы для поиска или обработки данных. Возможность использования этих специализированных возможностей не должна быть утеряна при доступе к данным через DiscoveryLink.

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

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

4. Выдача таких запросов и возврат результатов после их выполнения

Поскольку оболочки являются ключом к расширяемости в DiscoveryLink, одной из основных целей для архитектуры wrap-per было обеспечение реализации оболочек для максимально широкого спектра источников данных с минимальными усилиями. Чтобы сделать диапазон источников данных, к которым можно получить доступ с помощью DiscoveryLink, максимально широким, требуется только, чтобы источник данных (или приложение) имел некоторую форму программного интерфейса, который может отвечать на запросы и, как минимум, мог возвращать неотфильтрованные данные, смоделированные как строки таблицы. Автору оболочки не нужно реализовывать стандартный интерфейс запроса, который может быть слишком высокоуровневым или слишком низкоуровневым для базового источника данных. Вместо этого оболочка предоставляет информацию о возможностях обработки запросов источника данных и специализированных средствах поиска серверу DiscoveryLink, который динамически определяет, какую часть данного запроса источник данных способен обработать. Этот подход позволяет быстро создавать оболочки для простых источников данных, сохраняя при этом возможность использовать уникальные возможности обработки запросов нетрадиционных источников данных, таких как поисковые системы для химических структур или изображений. Оболочка может быть написана с минимальным знанием внутренней структуры DiscoveryLink. В результате стоимость написания базовой оболочки невелика. Оболочка, которая просто делает данные из нового источника доступными для DiscoveryLink, не пытаясь использовать большую часть собственных возможностей обработки запросов источника данных, может быть написана за считанные дни. Поскольку сервер DiscoveryLink может компенсировать отсутствующую функциональность в источниках данных, даже этот вид простой оболочки позволяет приложениям применять всю мощь SQL для извлечения новых данных и интеграции данных с информацией из других источников, хотя, возможно, и с производительностью ниже оптимальной. После написания базовой оболочки

ее можно постепенно улучшать, чтобы использовать больше возможностей обработки запросов источника данных, что приводит к повышению производительности и повышению функциональности, поскольку раскрываются специализированные алгоритмы поиска или другие новые возможности обработки запросов источника данных. Оболочка DiscoveryLink — это программа на языке C++, упакованная как общая библиотека, которая может динамически загружаться сервером DiscoveryLink при необходимости. Обычно одна оболочка способна получать доступ к нескольким источникам данных, если они используют общий или похожий интерфейс прикладного программирования (API). Это происходит потому, что оболочка не кодирует информацию о схеме, используемой в источнике данных. Таким образом, схемы могут развиваться без необходимости внесения изменений в оболочку, пока API источника остается неизменным. Например, оболочка Oracle, предоставляемая DiscoveryLink, может использоваться для доступа к любому количеству баз данных Oracle, каждая из которых имеет свою схему. Фактически, та же оболочка поддерживает несколько уровней релиза Oracle [2].

Ручная интеграция СУБД

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

Интеграция через Wrapper

Первый подход. Первым и обязательным условием является выбор СУБД для проектируемой гетерогенной системы баз данных. Этот выбор показывает решающее влияние на последующие процессы разработки и оценки систем для объединения данных SQL и NoSQL. В качестве СУБД SQL был выбран PostgreSQL. PostgreSQL — это объектно-реляционная система на основе данных с открытым исходным кодом, которая использует расширяемый язык SQL в сочетании с некоторыми функциями, которые позволяют безопасно хранить и масштабировать сложные рабочие данные. К основным преимуществам PostgreSQL можно отнести высокую степень развития: в этой СУБД есть поддержка главных объектов и их поведений, включая типы

данных, операции, функции, индексы и домены. По этой причине PostgreSQL можно назвать действительно гибким. Кроме того, эта СУБД предоставляет возможность создавать, хранить и использовать сложные структуры данных. Также стоит отметить, что PostgreSQL поддерживает вложенные и составные конструкции, которые не применяются и основаны на существующих стандартных реляционных базах данных. В качестве нереляционной СУБД была выбрана MongoDB — система управления базами данных NoSQL, которая набирает всю большую популярность на рынке и выделяет среди конкурентов своей технологией масштабировать потребление. К преимуществам MongoDB можно отнести гибкость, масштабируемость, доступность и высокую производительность.

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

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

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

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

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

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

- libbson;
- libmongoc;
- json-c.

Библиотеки libbson и libmongoc необходимы для корректной работы mongo_fdw, поскольку это расширение использует для работы драйвер языка C [3].

Разработка интегрированной схемы

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

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

Предварительный этап интеграции. Установление общего понимания существующих данных является предпосылкой успешной интеграции баз данных. Для этой цели входные схемы обычно преобразуются, чтобы сделать их максимально однородными. Исследователи в области интеграции баз

данных обычно предполагают, что все входные схемы выражены в одной и той же модели данных, так называемой «общей» модели данных (CDM). Этап перевода становится предпосылкой интеграции и рассматривается как отдельная проблема.

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

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

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

Было бы целесообразно создать интегрированные типы объектов, такие как Автор (соответственно Статья), представляющие всех авторов (соответственно все статьи) в любой библиотеке. Мы говорим, что два элемента (вхождение, значение, кортеж, ...) из двух баз данных соответствуют друг другу, если они описывают один и тот же факт реального мира (объект, связь или свойство).

Вместо того, чтобы определять соответствия экстенсионально, между экземплярами, соответствия определяются интенционально, между типами. Пример: каждая статья S2 соответствует статье S1, так что значение заголовка в S1 равно значению заголовка в S2. Интенсиональное определение называется утверждением соответствия между схемами (ICA). Процесс интеграции состоит в определении этих ICA и предоставлении для каждого из них федеративным пользователям глобального описания со всеми доступными данными по связанным элементам. Федеративная система хранит глобальное описание в интегрированной схеме (IS) и определение отображений между IS и локальными схемами. Полная интеграция существующих баз данных требует исчерпывающей идентификации и обработки всех соответствующих ICA. Тем не менее, возможно принять инкрементальный подход, при котором IS плавно обогащается по мере постепенного выявления новых соответствий [4].

Создание новой СУБД

Для решения задачи связи нескольких СУБД можно разработать совершенно новые СУБД.

hStorage-DB

Менеджер хранения СУБД обычно является интерфейсом для преобразования запроса данных СУБД в запрос ввода-вывода. В процессе преобразования вся семантическая информация удаляется, оставляя только информацию о физической компоновке запроса: логический адрес блока, направление (чтение/запись), размер и фактические данные, если это запись.

Это, по сути, создает семантический разрыв между СУБД и системами хранения. В hStorage-DB преодолевается семантический разрыв, предоставляя выбранную и важную семантическую информацию менеджеру хранения, который, следовательно, может классифицировать запросы на разные типы.

С набором предопределенных правил каждый тип ассоциируется с политикой качества обслуживания (QoS), которую может поддерживать базовая система хранения.

Во время выполнения, используя протокол дифференцированных услуг хранения, связанная политика запроса передается в систему хранения вместе с самим запросом. Получив запрос, система хранения сначала извлекает связанную политику QoS, а затем использует соответствующий механизм для обслуживания запроса в соответствии с требованиями политики QoS.

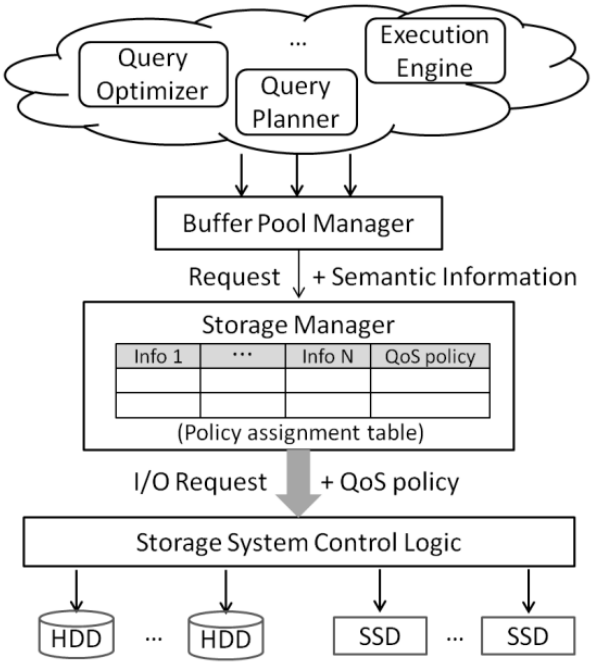

Рисунок 1. Архитектура hStorage-DB

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

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

hStorage-DB основана на PostgreSQL 9.0.4. В основном это касается трех вопросов: (1) Она оснащена оптимизатором запросов и механизмом выполнения для извлечения семантической информации, встроенной в деревья планов запросов и в запросы пула буферов. (2) Имеет расширенную структуру данных пула буферов для хранения собранной семантической информации. Менеджер хранилища также был расширен для включения «таблицы

назначения политик». (3) Наконец, поскольку PostgreSQL является многопроцессорной СУБД, для работы с параллелизмом был выделен небольшой регион общей памяти для глобальных структур данных, к которым должны иметь доступ все процессы.

Ключом к эффективности hStorage-DB является связывание каждого запроса с надлежащей политикой QoS [5].

FINDIT

Авторы в статье [6] представляют двухуровневый подход, чтобы преодолеть гетерогенность и учитывать автономию баз данных насколько это возможно, и при этом концептуально построить Гиперраспределенную Базу Данных на основе существующих баз данных. Пользователи постепенно и динамически обучаются доступному информационному пространству, не будучи перегруженными всей доступной информацией. Двухуровневая структура предоставляет участвующим базам данных гибкое средство для обмена информацией. Система, которая реализует этот подход, называется FINDIT.

Двухуровневый подход, который предлагается соответствует коалициям (первый уровень) и сервисным ссылкам (второй уровень). Коалиции являются средством для баз данных, чтобы быть сильно связанными, в то время как сервисные ссылки являются средством для них, чтобы быть слабо связанными. Со-базы данных вводятся как средство для реализации этих концепций и как помощь для межсайтового обмена данными.

Коалиции являются группировками баз данных, которые разделяют некоторый общий интерес. Это состоит в интересе к информационному мета-типу (например, Рестораны). В этом контексте базы данных будут делиться описаниями этого типа информации.

Во многих странах политические партии, придерживающиеся разных идеологий, согласны на минимальный набор целей на ограниченный период времени. Результатом является коалиция. Таким образом, коалиции основаны на краткосрочном интересе. В политических коалициях члены сохраняют свою

автономию, оставаясь при этом приверженными набору правил, которые были согласованы. Наша концепция коалиций поэтому очень близка к концепции политических коалиций. С другой стороны, концепция федерации больше похожа на федерацию штатов, где набор правил является долгосрочным и где штаты пользуются разумным, хотя и ограниченным, объемом автономии.

Другим примером группировки, близким к концепции коалиций, является Интернет. Интернет — это компьютерная сеть, которая состоит из подсетей, которые соединены друг с другом. Каждый подсеть имеет свой собственный набор стандартных протоколов для общения. Хотя, в отличие от FINDIT, все подсети предоставляют почти одинаковый набор информации, идея кооперативной среды существует. В каждой подсети участвующие сайты подчиняются набору правил, которые регулируют общение между собой. Подсети обычно создаются для выполнения определенной цели в рамках некоторых географических границ. Например, NSFnet — это сеть, целью которой является связывание крупных исследовательских учреждений в США (географическая граница) для проведения исследований с использованием суперкомпьютеров (цель). Другим примером подсети является DECnet, где границей является компания (организационная граница), а целью является предоставление структуры для их исследователей для обмена информацией. Каждая сеть связана с другими сетями, в основном для обмена электронной почтой.

Сервисные ссылки — это упрощенный способ для баз данных делиться информацией. Они позволяют обмениваться с низкими накладными расходами. Сервисные ссылки обычно предполагают некоторые обязательства, которые похожи на контракты. В терминах обмена данными, объем, ожидаемый для обмена в сервисной ссылке, обычно включает минимальное количество информации. Например, ожидается, что будет поделено только имя информации с несколькими синонимами и информация о базе данных, которая экспортирует эту информацию. В этом отношении сервисные ссылки несут низкие накладные расходы по сравнению с использованием коалиций.

Сервисные ссылки могут возникать между любыми двумя сущностями. Сервисные ссылки могут быть только одного из трех типов. Первый тип включает сервисную ссылку между двумя коалициями для обмена информацией. Второй тип включает сервисную ссылку между двумя базами данных. Третий тип включает сервисную ссылку между коалицией и базой данных. Сервисная ссылка между двумя коалициями включает предоставление общего описания информации, которая будет делиться. Аналогично, сервисная ссылка между двумя базами данных также включает предоставление общего описания информации, которую базы данных хотели бы поделиться. Третья альтернатива — это сервисная ссылка между коалицией и базой данных. В этом случае база данных (или коалиция) предоставляет общее описание информации, которую она готова поделиться с коалицией (или базой данных). Разница между этими тремя альтернативами заключается в том, как разрешаются запросы. В первой и третьей альтернативе (когда поставщик информации — это коалиция) предоставляющая коалиция берет на себя дальнейшее разрешение запроса. Во втором случае, однако, пользователь отвечает за контакт с администратором предоставляющей базы данных, чтобы узнать больше о информации.

Когда базы данных неохотно показывают слишком много деталей о типе информации, которую они содержат, у них есть выбор присоединиться к сервисной ссылке с другими базами данных или коалициями. По сути, сервисные ссылки — это средства для баз данных быть слабо связанными с другими базами данных (или коалициями). Это предоставляет структуру, в которой базы данных обмениваются минимальным количеством данных о информации, которую они хотели бы поделиться. Обмениваются только общая информация о фактической информации, которую нужно поделиться, и информация о обслуживающих базах данных. Следует подчеркнуть, что, как и в любой сервисной ссылке, существует услуга, которую должна предоставить одна из вовлеченных сущностей, т.е. коалиции или базы данных.

Со-базы данных — это объектно-ориентированные базы данных,

прикрепленные к каждой участвующей базе данных. Объектно-ориентированность зарекомендовала себя как отличная парадигма моделирования для сложной структуры и поведения. Наследование и инкапсуляция оказались критически важными при проектировании надежных и легко поддерживаемых программных систем. Схема со-базы данных состоит из двух подсхем. Каждая подсхема представляет либо коалицию, либо сервис. Каждая подсхема состоит из решетки классов. Каждый класс представляет набор баз данных, которые могут отвечать на запросы о конкретном типе информации (например, запросы о ресторанах). Подсхема сервиса состоит из двух подсхем: первая — это подсхема услуг, которые коалиции, членом которых она является, имеет с другими базами данных и коалициями, а вторая — это подсхема услуг, которые база данных имеет с другими базами данных и коалициями. Каждая из этих подсхем, в свою очередь, состоит из двух подклассов, которые соответственно описывают услуги с базами данных и услуги с другими коалициями. Подсхема коалиции состоит из одной или нескольких подсхем, где каждая из них представляет коалицию.

Описание о коалиционных сервисах включает информацию о точках входа и контакте с этими коалициями. Другие описания предоставляют информацию местным базам данных, чтобы можно было выбрать лучшую точку контакта. Следует отметить, что подсхема, представляющая набор коалиционных сервисов, будет одинаковой для всех баз данных, которые являются членами обслуживающей коалиции.

Унифицировать все имеющиеся СУБД

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

UDBMS

Фундаментальную модель унифицированной системы управления данными можно проследить до системы управления объектно-реляционными базами данных (ORDBMS), которая интегрирует объектно-ориентированные функции в реляционную модель. Система ORDBMS может управлять

различными типами данных, такими как реляционные, объектные, текстовые и пространственные, подключая доменно-специфические типы данных, функции и реализации индексов в ядра СУБД. Например, PostgreSQL поддерживает реляционные, пространственные и XML-данные. Oracle продолжает усилия OR по поддержке XML, JSON и графовых данных.

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

Такая абстракция может скрыть детали реализации данных от пользователей и облегчить глобальный доступ и запрос для различных типов данных. Текущие усилия авторов статьи [7] в этом направлении направлены на объединение пяти типов данных, включая отношение, ключ-значение, JSON, XML и граф. Эта цель может быть достигнута в два этапа. На первом этапе представляем гибкий способ представления графа, JSON, XML и моделей ключ-значение в виде унифицированной модели NoSQL (UNM) логически. На втором этапе будут исследованы новые подходы для объединения UNM и моделей отношений. В конечном итоге унифицированная модель может

поддерживать все пять типов данных.

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

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

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

Глобальная база данных

В статье [8] для решения данной проблемы предлагается 4-уровневой архитектуре клиент-сервер с использованием системы с несколькими базами данных можно визуализировать как систему клиент-сервер, которая позволяет клиентам одновременно получать доступ и обновлять данные, хранящиеся на нескольких серверах распределенных баз данных:

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

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

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

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

Существует клиентская программа, которая находится на верхнем уровне глобальной системы управления базами данных и глобальной схемы сервера многобазовой системы. Глобальная схема создается с набором виртуальных глобальных классов и хранится в глобальной базе данных (GDB). Пользователь будет отправлять запрос на глобальную схему, используя веб-интерфейс или графический пользовательский интерфейс программы приложения, запрос будет разбит на набор подзапросов и будет отправлен на соответствующие удаленные локальные компонентные серверы баз данных и будет выполняться локально.

Примером реализации данной архитектуры может являться MOMIS [9].

Модификация JSON

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

Тем не менее, характеристики самих реляционных баз данных привели к их неотъемлемой неспособности выполнять обработку JSON. Этот недостаток также заставил разработчиков неохотно использовать единую реляционную

базу данных для одновременной обработки высокопроизводительных данных и сложных логических реляционных данных в современной гибридной облачной системе хранения.

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

Авторами [10] предложен гибридный язык запросов JSON на основе SQL. Были предложены и сравнены два различных метода отображения, которые использовались для хранения данных JSON в реляционных базах данных.

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

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

С точки зрения исследования сопоставления данных JSON с реляционными данными был предложен алгоритм сопоставления из JSON в реляционную базу данных, и данные JSON были сохранены в реляционной базе данных. JSON был определен в веб-запросе данных, и был проведен теоретический анализ для изучения метода ограничений целостности JSON.

Была предложена формальная модель данных JSON и был определен легкий язык запросов. Был предложен формат обмена данными между службами RESTFul, который больше склонен хранить данные сетевых атрибутов.

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

Взаимодействие в виде графа

В статье [11] авторы предлагают глобальную систему со следующими основными характеристиками:

- Графовый формализм (а именно GM) для представления и запроса баз данных. Этот формализм подходит для придания точной семантики сложным визуальным представлениям и является достаточно общим для формализации, в принципе, базы данных, выраженной в любой из наиболее распространенных моделей данных. Примитивы запроса формализма, хотя и состоят исключительно из двух элементарных графических действий, а именно выбора узла и рисования ребра, по крайней мере столь же выразительны, как реляционная алгебра.
- Адаптивный визуальный интерфейс, построенный на основе вышеуказанного формализма, предоставляющий пользователю различные визуальные представления и механизмы взаимодействия вместе с возможностью переключения между ними. Все различные визуальные представления позволяют выразить по крайней мере класс конъюнктивных запросов.
- Определение трех подходящих наборов алгоритмов перевода, один для

перевода базы данных, выраженной в любой из наиболее распространенных моделей данных, во внутреннюю системную модель, один для перевода запроса GM в терминах языков запросов базовых моделей данных и один, предназначенный для реализации последовательного переключения между различными визуальными представлениями во время формулировки запроса.

- Построение и управление эффективной пользовательской моделью, которая позволяет системе предоставлять пользователю наиболее подходящее визуальное представление в соответствии с его навыками и потребностями.

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

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

В более поздних предложениях интенсиональная и экстенсиональная части базы данных сосуществуют. Диаграммные представления являются

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

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

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

Выводы

В ходе исследования были рассмотрены различные подходы к решению проблемы взаимодействия между СУБД. В работе были выделены и в последствии были описаны следующие группы методов: узконаправленные системы, ручная интеграция СУБД, создание новой СУБД, унифицировать имеющиеся СУБД, взаимодействие в виде графа.

Список использованных источников

1. When will we have true heterogeneous database systems / A. P. Sheth; — In Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow (ACM '87). — IEEE Computer Society Press, Washington, DC, USA, 1987. — 747–748 с.
2. DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources / Laura Haas, Peter Schwarz, Prasad Kodali, Elon Kotlar, Julia Rice, William Swope; — IBM Systems Journal, vol. 40, 2001. — 489–511 с. — DOI: 10.1147/sj.402.0489.
3. The Study of Combining SQL and NoSQL Databases in a Heterogeneous System for the Development of a Project Management Database / O.V. Shalina, E.E. Andrianova; — Peter the Great St. Petersburg Polytechnic University; — Theoretical & Applied Science, 2024, no. 5 (133). — pp. 14-17.
4. Issues and Approaches of Database Integration / Christine Parent, Stefano Spaccapietra; — Communications of the ACM, vol. 41, no. 5es, May 1998. — pp. 166–178. — DOI: 10.1145/276404.276408.
5. hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems / Tian Luo, Rubao Lee, Michael Mesnier, Feng Chen, Xiaodong Zhang; — Proceedings of the VLDB Endowment, vol. 5, 2012.

6. On Building a Hyperdistributed Database / Michael Papazoglou, Athman Bouguettaya; — Information Systems, vol. 20, no. 5, 1995. — DOI: 10.1016/0306-4379(95)00030-8.
7. UDBMS: Road to Unification for Multi-model Data Management / Jiaheng Lu, Zhen Liu, Pengfei Xu, Chao Zhang; — In: Advances in Databases and Information Systems (ADBIS), 2018. — 330–344 c. — DOI: 10.1007/978-3-030-01391-2_33.
8. A Multidatabase System as 4-Tiered Client-Server Distributed Heterogeneous Database System / Ali Ghulam; — International Journal of Computer Science and Information Security, vol. 6, 2009. — 010–014 c. — DOI: 10.48550/arXiv.0912.0579.
9. MIKS: An Agent Framework Supporting Information Access and Integration / Domenico Beneventano, Sonia Bergamaschi, Gionata Gelati, Francesco Guerra, Maurizio Vincini; — Dipartimento Ingegneria dell'Informazione, Università di Modena, via Vignolese 905, 41100 Modena, Italy; CSITE-CNR, viale Risorgimento 2, 40136 Bologna, Italy.
10. JSON-Based Control Model for SQL and NoSQL Data Conversion in Hybrid Cloud Database / Lei Zhang, Ke Pang, Jiangtao Xu, Bingxin Niu; — Journal of Cloud Computing, vol. 11, 2022. — DOI: 10.1186/s13677-022-00302-9.
11. Graphical Interaction with Heterogeneous Databases / T. Catarci, G. Santucci, J. Cardiff; — The VLDB Journal, vol. 6, 1997. — 97–120 c. — DOI: 10.1007/s007780050035.