MySQL 8 для больших данных

Среди организаций, работающих с крупными объемами данных на регулярной основе, реляционная система управления базами данных MySQL стала популярным решением по обработке структурированных больших данных.
В этой книге вы познакомитесь с тем, как администраторы баз данных могут использовать MySQL для обработки миллиардов записей и извлечения данных с производительностью, сравнимой или превосходящей коммерческие решения для СУБД с более высокими затратами.
Показано как реализовывать успешную стратегию больших данных с помощью таких технологий как Apache Hadoop, MapReduce и MySQL Applier.
Кроме того, книга включает в себя прктические примеры использования Apache Sqoop для обработки событий в режиме реального времени.

Прочитав книгу, вы:

- познакомитесь с особенностями MySQL 8 и тем, как их можно использовать для обработки больших данных;
- откроете новые возможности MySQL 8 по управлению структурированными и неструктурированными большими данными;
- интегрируете MySQL 8 и Hadoop с целью эффективной обработки данных;
- выполните агрегацию с использованием MySQL 8 для оптимального использования данных;
- изучите различные виды соединений и объединений в MySQL 8 для эффективной обработки больших данных;
- ускорите обработку больших данных с помощью Memcached;
- интегрируете MySQL с API NoSQL;
- реализуете репликацию для создания высокодоступных решений для больших данных.

ISBN 978-5-97060-653-7

Интернет-магазин:
www.dmkpress.com
Книга-почтой:
orders@alians-kniga.ru
Оптовая продажа:
«Альянс-книга»
(499)782-3889
books@alians-kniga.ru

ДМК ИЗДАТЕЛЬСТВО
www.дмк.рф

Шаббир Чаллавала
Джадип Лакхатария
Чинтан Мехта
Кандарп Патель

MySQL 8 для больших данных

MySQL 8
для больших данных

Эффективная обработка данных с помощью MySQL 8, Hadoop, API NoSQL и других инструментов для больших данных

Packt

ДМК ИЗДАТЕЛЬСТВО

Шаббир Чаллавала, Джадип Лакхатария,
Чинтан Мехта, Кандарп Патель

MySQL 8 для больших данных

Shabbir Challawala, Jaydip Lakhatariya,
Chintan Mehta, Kandarp Patel

MySQL 8 for Big Data

Effective data processing with MySQL 8, Hadoop, NoSQL APIs, and other Big Data tools

Packt>
BIRMINGHAM – MUMBAI

Шаббир Чаллавала, Джадип Лакхатария,
Чинтан Мехта, Кандарп Патель

MySQL 8 для больших данных

Эффективная обработка данных с помощью MySQL 8, Hadoop, NoSQL API и других инструментов для больших данных

Москва, 2018

УДК 004.65
ББК 32.973.26
Ч12

Чаллавала Ш., Лакхатария Дж., Мехта Ч., Патель К.
Ч12 MySQL 8 для больших данных / пер. с анг. А. В. Логунова. – М.: ДМК Пресс, 2018. – 226 с.: ил.

ISBN 978-5-97060-653-7

Среди организаций, работающих с крупными объемами данных на регулярной основе, реляционная система управления базами данных MySQL стала популярным решением по обработке структурированных больших данных. В книге вы познакомитесь с тем, как администраторы баз данных могут использовать MySQL для обработки миллиардов записей и извлечения данных с производительностью, сравнимой или превосходящей коммерческие решения для СУБД с более высокими затратами. Показано, как реализовывать успешную стратегию больших данных с помощью таких технологий, как Apache Hadoop, MapReduce и MySQL Applier. Также книга включает в себя практические примеры использования Apache Sqoop для обработки событий в режиме реального времени.

Издание будет полезно администраторам баз данных MySQL и специалистам по большим данным, которые хотят интегрировать MySQL и Hadoop с целью реализации высокопроизводительных решений.

УДК 004.65
ББК 32.973.26

ISBN 978-1-78839-718-6 (анг.)
ISBN 978-5-97060-653-7 (рус.)

Содержание

Об авторах

Шаббир Чаллавала имеет более чем 8-летний богатый опыт в предоставлении решений на основе технологий MySQL и PHP. В настоящее время он работает с KNOWARTH Technologies. Он принимал участие в разработке предназначенных для предприятий различных PHP-решений в области электронной коммерции и обучающих порталов; работал с различными вычислительными платформами на основе PHP, такими как Magento E-commerce, Drupal CMS и Laravel.

Шаббир участвовал в различных корпоративных решениях на разных этапах их реализации, в частности проектировании архитектуры, оптимизации баз данных и настройки производительности, и отлично разбирается в жизненном цикле разработки программного обеспечения. Он работал над интеграцией технологий больших данных, в частности MongoDB и Elasticsearch с платформой PHP.

Я искренне благодарен Чинтан Мехта, который оказал мне доверие, поручив написание этой книги. Я хотел бы поблагодарить KNOWARTH Technologies за предоставленную возможность и поддержку стать частью данной книги. Я также хочу поблагодарить моих соавторов и команду Packt Publishing за замечательную поддержку. Особенно хотел бы поблагодарить моих маму, папу, жену Сакину, прекрасного сына Мохаммада и членов семьи за поддержку на протяжении всего проекта.

Джадип Лакхатария имеет богатый опыт работы в портальных и J2EE-платформах, быстро адаптируется к любой новой технологии и неуклонно работает над совершенствованием своих знаний. В настоящее время Джадип связан с ведущей компанией по внедрению открытого исходного кода на предприятиях, KNOWARTH Technologies (www.knowarth.com), где занимается различными предпринимательскими проектами.

Джадип, являясь полностековым разработчиком, доказал свою универсальность благодаря освоению таких технологий, как Liferay, Java, Spring, Struts, Hadoop, MySQL, Elasticsearch, Cassandra, MongoDB, Jenkins, SCM, PostgreSQL, и многих других.

Он был награжден премиями, в частности за заслуги, приверженность службе, а также в качестве звездного исполнителя. Он любит выступать наставником и проводит курсы по темам порталов и платформ J2EE.

Я искренне благодарен моим прекрасным соавторам, и особенно г-ну Чинтан Мехта, за мотивацию и веру в меня. Хотел бы поблагодарить KNOWARTH за постоянное предоставление новых возможностей самосовершенствования. Хотел бы также поблагодарить всю команду Packt Publishing за замечательную поддержку на протяжении всего проекта.
Наконец, я очень благодарен моим родителям и младшему брату Кейуру за то, что они поддерживали меня на протяжении всего путешествия. Спасибо моим друзьям и коллегам за поддержку.

Чинтан Мехта является соучредителем KNOWARTH Technologies (www.knowarth.com) и возглавляет направление Cloud/RIMS/DevOps. Он имеет богатый про-

грессивный опыт в операционных системах, администрировании серверов Linux, AWS Cloud, DevOps, RIMS и администрировании серверов на технологиях с открытым исходным кодом. Он также является сертифицированным архитектором решений AWS.

На протяжении своей карьеры в области инфраструктуры и операций жизненно важная роль Чинтан также проявлялась в анализе требований, проектировании архитектуры и безопасности, планировании высокой доступности и аварийного восстановления, автоматизированном мониторинге и развертывании, помощи клиентам в процессах сборки, настройки производительности, развертывании и настройки инфраструктуры, а также в настройке и развертывании приложений. Он также отвечал за создание различных офисов в разных местах, с фантастической единоличной ответственностью за достижение оперативной готовности для организаций, с которыми он был связан.

У своего предыдущего работодателя он возглавлял отдел управляемых облачных служб и получил множество премий в знак признания высокоценного вклада в деятельность группы. Он также возглавлял команды внедрения ISO 27001:2005 в качестве представителя совместного руководства компании. Чинтан является автором книги «*Решения для резервного копирования и восстановления Hadoop*» (Hadoop Backup and Recovery Solutions), а также был рецензентом книг «*Рекомендации по повышению производительности портала Liferay*» (Liferay Portal Performance Best Practices) и «*Создание бессерверных веб-приложений*» (Building Serverless Web Applications).

Он обладает дипломом по компьютерному оборудованию и сетям известного в Индии института.

> *При написании этой книги я опирался на многих людей, как прямо, так и косвенно. Прежде всего хотел бы поблагодарить моих соавторов и замечательную команду PacktPub за свои усилия. Я хотел бы особенно поблагодарить мою замечательную жену, Миттал, и моего милого сына, Девам, за то, что они с достоинством вытерпели все те долгие дни, ночи и выходные, когда я ночевал в своем ноутбуке. Многие люди вдохновили и внесли свой вклад в эту книгу и предоставили комментарии, правки, мысли и идеи, в особенности Крупал Кхатри и Чинтан Гаджар. Ничто не могло помешать моей книге. Я также хочу поблагодарить всех рецензентов этой книги.*
>
> *И последнее, но не менее важное: я хочу поблагодарить моих маму и папу, друзей, семью и коллег за поддержку на протяжении всего времени написания данной книги.*

Кандарп Патель возглавляет направление PHP в KNOWARTH Technologies (www.knowarth.com). Он обладает обширным опытом в обеспечении сквозных решений в CMS, LMS, WCM и электронной коммерции, а также во внедренческих проектах, связанных с различными интеграциями, для корпоративных клиентов. Его богатый опыт в предоставлении решений с использованием MySQL, MongoDB и платформ на основе PHP насчитывает более чем 9 лет. Кандарп также является сертифицированным разработчиком MongoDB и Magento.

Кандарп обладает опытом разработки корпоративных приложений на различных этапах жизненного цикла разработки программного обеспечения и играет важную роль в сборе требований, разработке архитектуры, разработке баз данных, разработке приложений, настройке производительности и CD/CI.

Кандарп имеет степень бакалавра технических наук в области информационных технологий из известного в Индии университета.

Хотел бы отметить: Чинтан Мехта вел меня через различные этапы американских горок при написании книги. Хотел бы поблагодарить KNOWARTH Technologies за предоставленную мне возможность стать частью этой книги. Кроме того, хотел бы поблагодарить моих великолепных соавторов и команду PacktPublishing за замечательную поддержку на протяжении всего путешествия.

Последнее, но не маловажное, я хочу поблагодарить моих маму и папу, и мою жену Халапа, за постоянную поддержку и поощрения во время написания этой книги. Я посвящаю свою первую книгу моим прекрасным принцессам, Джейне и Джайсви.

О рецензентах

Анкит Бхавсар является старшим консультантом KNOWARTH Technologies (www.knowarth.com) и возглавляет команду, работающую над решениями для планирования корпоративных ресурсов. Он обладает богатыми познаниями Java, JEE, MySQL, PostgreSQL, Apache Spark и многих других инструментов и технологий с открытым исходным кодом, используемых при создании приложений корпоративного уровня.

В своей карьере Анкит решал разные задачи – как архитектор приложения и программист, архитектор структуры баз данных для портала Astrology, включая объектно-ориентированное программирование, технический архитектурный анализ, проектирование и разработку, а также проектирование, развитие, совершенствование и обработку баз данных, моделирование данных и объектов в широком спектре приложений и сред для обеспечения технических и бизнес-решений для клиентов.

Анкит имеет степень магистра по компьютерным приложениям Университета Северного Гуджарата.

Прежде всего хотел бы поблагодарить моих рецензентов и замечательную команду Packt Publishing за их усилия. Я также хотел бы поблагодарить Субхаш Шаха и Чинтан Мехта. Еще хочу поблагодарить всех авторов этой книги. И последнее, но не менее важное: хочу поблагодарить мою маму, друзей, семью и коллег за поддержку на протяжении всего рецензирования этой книги.

Гаджар Чинтан является консультантом по технологиям в KNOWARTH (www.knowarth.com). Он имеет богатый передовой опыт в JavaScript, NodeJS, BackboneJS, Angularjs, Java и MongoDB, а также предоставляет услуги корпоративного уровня, в частности услуги в области разработки корпоративных порталов, внедрения ERP и интеграции предприятий с привлечением технологий с открытым исходным кодом.

На протяжении всей своей карьеры в сфере корпоративных услуг Чинтан также играл жизненно важную роль в оказании помощи клиентам по вопросам анализа требований, архитектурному дизайну, реализации пользовательского интерфейса и процесса сборки, следуя лучшим практическим рекомендациям по разработке и процессам с доведением порученных проектов до стадии развертывания и реализации идей клиента и организаций, с которыми он был связан.

На протяжении своей карьеры Чинтан активно участвовал в развитии корпоративных решений, связанных с планированием ресурсов, работал над разработкой одностраничного приложения (SPA), а также над мобильным приложением с привлечением NodeJS, MongoDB и AngularJS. Работа Чинтана получила большое признание за весьма ценный вклад, внесенный в команду и компанию в целом. Чинтан участвовал в написании книги «*Решения для резервного копирования и восстановления Hadoop*» (Hadoop Backup and Recovery Solutions). Он имеет степень магистра по компьютерным приложениям (CMA) университета Ганпат.

Хотел бы поблагодарить основных сорецензентов и замечательную команду Packt Publishing за их усилия. Также хотел бы поблагодарить Субхаш Шаха, Чинтан Мехту и Анкита Бхавсара и его коллег за поддержку, оказанную мне в ходе рецензирования этой книги. Благодарю всех авторов этой книги.

Никундж Ранпура обладает богатым передовым опытом в операционных системах и администрировании серверов Linux, AWS Cloud, Devops, RIMS, сетях, системах хранения данных, резервном копировании, обеспечении безопасности и администрирования серверов на основе технологий с открытым исходным кодом. Он быстро адаптируется к любой технологии и имеет острое желание постоянного совершенствования. Он также является сертифицированным архитектором решений AWS.

В своей карьере Никундж выступал в различных ролях, в том числе как системный аналитик, ИТ-менеджер, руководитель направления управляемых облачных служб, архитектор инфраструктуры, разработчик инфраструктуры, архитектор DevOps, архитектор AWS и менеджер поддержки для различных крупных реализаций. Он принимал участие в создании решений и консультировании по созданию служб SaaS, IAAS и PAAS в облаке.

В настоящее время Никундж связан с ведущей компанией по внедрению открытого исходного кода на предприятиях, KNOWARTH Technologies, в качестве ведущего консультанта, где он занимается корпоративными проектами, помогая клиентам в отношении анализа требований, проектирования архитектуры, проектирования безопасности, высокой доступности и планирования аварийного восстановления, а также в руководстве командой.

Никундж окончил Университет Бхавнагара и прошел сертификацию по межсетевым экранам CISCO и UTM. Он был отмечен двумя наградами за свой ценный вклад в компанию. Он также является участником Stack Overflow. С ним можно связаться по адресу ranpura.nikunj@gmail.com.

Хотел бы поблагодарить мою семью за их огромную поддержку и веру в меня на протяжении всей моей стадии обучения. Мои друзья выразили ко мне такое доверие, которое заставляет меня делать лучшее, на что я способен. Я счастлив, что Бог благословил меня такими замечательными людьми, которые меня окружают и без которых мой сегодняшний успех был бы невозможен.

Субхаш Шах является архитектором программного обеспечения с более чем 11-летним опытом разработки веб-ориентированных программных решений на основе различных платформ и языков программирования. Он является энтузиастом объектно-ориентированного программирования и ярым сторонником разработки свободного программного обеспечения с открытым исходным кодом, а также его использования предприятиями для снижения риска, уменьшения затрат и обеспечения большей гибкости. Его карьерные интересы включают проектирование устойчивых программных решений. Лучшие из его технических навыков не ограничиваются анализом требований, проектированием архитектуры, мониторингом доставки проекта, настройкой приложений и инфраструктуры и настройкой процесса выполнения. Он является поклонником написания качественного кода и тестирования.

Субхаш работает главным консультантом в KNOWARTH Technologies Pvt Ltd. и возглавляет ERP-направление. Он получил степень бакалавра в области информационных технологий Университета Северного Гуджарата, Хемчандрачарья.

Приятно выступать рецензентом этой книги. Хотел бы поблагодарить команду Packt Publishing за предоставление такой возможности. Хотел бы поблагодарить мою семью за поддержку на протяжении рецензирования этой книги. Было бы трудно, если бы они не понимали мои приоритеты и не были источником вдохновения. Хочу поблагодарить коллег за постоянную поддержку и помощь. Наконец, хочу поблагодарить авторов за написание такого полезного и подробного контента.

Предисловие

Среди организаций, обрабатывающих крупные объемы данных на регулярной основе, реляционная система управления базами данных MySQL стала популярным решением для работы со *структурированными большими данными*. В этой книге вы познакомитесь с тем, как **Администраторы баз данных** (**АБД**) могут использовать MySQL для обработки миллиардов записей и загрузки и извлечения данных с производительностью, сравнимой или превосходящей коммерческие решения для СУБД с более высокими затратами.

Многие организации сегодня зависят от MySQL для своих веб-сайтов и решений по обработке больших данных в плане своих потребностей в архивировании, хранении и анализе данных. Однако их интеграция может оказаться сложной задачей. Эта книга покажет, как реализовывать успешную стратегию больших данных с помощью Apache Hadoop, вычислительной платформы для разработки и выполнения распределенных программ и MySQL 8. В ней будут рассмотрены варианты сценариев использования в режиме реального времени, которые объяснят способы интеграции и достижения решений по обработке больших данных с использованием различных технологий, таких как Apache Hadoop, Apache Sqoop и MySQL Applier.

В книге, в частности, будут рассмотрены такие темы, как особенности MySQL 8, практические рекомендации по использованию MySQL 8 и API NoSQL, предоставляемого этой реляционной СУБД, а также будет приведен пример использования MySQL 8 для управления большими данными. В конце этой книги вы узнаете, как эффективно использовать MySQL 8 в целях управления конкретными данными приложений, предназначенных для обработки больших данных.

О чем эта книга рассказывает

Глава 1 «*Введение в большие данные и MySQL 8*» содержит обзор больших данных и MySQL 8, их важность и жизненный цикл больших данных. В ней рассматривается основной принцип больших данных и их тенденции на текущем рынке. Наряду с этим эта глава также посвящена объяснению преимуществ использования MySQL, она проведет нас в пошаговом режиме по процессу инсталляции MySQL 8 и познакомит с недавно введенными в MySQL 8 функциональными средствами.

Глава 2 «*Методы запроса данных в MySQL 8*» посвящена основам запросов данных в MySQL 8 и тому, как соединять или агрегировать в ней набор данных.

Глава 3 «*Индексирование данных для высокопроизводительных запросов*» подробно объясняет индексирование в MySQL 8, вводит различные типы индексирования, которые имеются в MySQL, и показывает, как выполнять индексирование для более быстрой работы на крупных объемах данных.

Глава 4 «*Использование Memcached с MySQL 8*» предоставляет обзор работы с Memcached в MySQL и информирует о различных преимуществах использования этого плагина. В ней рассматриваются шаги установки Memcached, конфигурирование репликации и различные API Memcached на разных языках программирования.

Глава 5 «*Разделение данных больших объемов*» объясняет, каким образом в MySQL 8 можно разделять крупные объемы данных, используя различные методы разделения. Она охватывает различные типы разделения, которые можно реализовать в MySQL 8, и их использование с большими данными.

Глава 6 «*Репликация для построения высокодоступных решений*» объясняет реализацию групповой репликации в MySQL 8. В главе рассказывается о том, каким образом большие данные можно масштабировать и как репликация данных может становиться быстрее с использованием различных методов репликации.

Глава 7 «*Практические рекомендации по работе с MySQL 8*» посвящена лучшим приемам использования MySQL 8 для больших данных. Она содержит разнообразные советы в отношении того, что можно и чего нельзя делать при использовании MySQL 8.

Глава 8 «*Прикладной программный интерфейс NoSQL для интегрирования с решениями для больших данных*» объясняет интеграцию API NoSQL для получения данных. В ней также даются пояснения в отношении технологии NoSQL и ее различных API на разных языках программирования для соединения NoSQL с MySQL.

Глава 9 «*Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop*» объясняет, как массовые данные могут эффективно передаваться между Hadoop и MySQL с помощью Apache Sqoop.

Глава 10 «*Практический пример: часть II. Обработка событий в реальном времени с помощью MySQL Applier*» объясняет интеграцию MySQL в режиме реального времени с Hadoop, чтение событий двоичного журнала сразу после их фиксации и запись их в файл в распределенной файловой системе HDFS.

Что требуется для этой книги

Эта книга послужит для вас гидом по установке всех инструментов, которые вам потребуются, чтобы проследить работу приводимых примеров. Для эффективного выполнения примеров кода, представленных в данной книге, необходимо установить следующее программное обеспечение:

- MySQL 8.0.3;
- Hadoop 2.8.1;
- Apache Sqoop 1.4.6.

Для кого эта книга предназначена

Эта книга предназначена для администраторов баз данных MySQL и профессиональных специалистов по большим данным, которые хотят интегрировать MySQL и Hadoop с целью реализации высокопроизводительного решения по обработке больших данных. Некоторый предыдущий опыт работы с реляционной СУБД MySQL будет полезен.

Условные обозначения

В этой книге вы найдете ряд текстовых стилей, которые выделяют различные виды информации. Вот некоторые примеры этих стилей и объяснение их значения.

Кодовые слова в тексте, имена таблиц баз данных, папок, файлов, расширения файлов, пути, фиктивные URL-адреса, ввод данных пользователем и дескрипторы

социальной сети Twitter показаны следующим образом: «И уже установленные пакеты могут быть обновлены при помощи флага include».

Фрагмент исходного кода оформляется следующим образом:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Когда мы хотим обратить ваше внимание на определенную часть фрагмента кода, соответствующие строки или элементы выделены жирным шрифтом:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Любой ввод или вывод командной строки записывается следующим образом:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
/etc/asterisk/cdr_mysql.conf
```

Новые термины и важные слова выделены жирным шрифтом. Слова, которые вы видите на экране, например в меню или диалоговых окнах, появляются в тексте следующим образом: «Нажатие кнопки **Next** (Далее) переводит вас на следующий экран».

Предупреждения или важные примечания отображаются в этом поле.

Советы и приемы появляются тут.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или вебсайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Введение в большие данные и MySQL 8

Сегодня мы живем в эпоху цифровизации. Мы производим огромное количество данных по многим направлениям: социальные сети, покупки в продуктовых магазинах, транзакции по банковским/кредитным картам, электронные письма, хранение данных в облаках и т. д. Один из первых вопросов, который приходит на ум: получаете ли вы максимальную отдачу от собранных данных? Для этого цунами данных мы должны иметь соответствующие инструменты, чтобы можно было получать эти данные в организованном порядке, который мог бы использоваться в различных областях, таких как научные исследования, передача данных в режиме реального времени, борьба с преступностью, обнаружение мошенничества, цифровая персонализация и т. д. Все эти данные должны фиксироваться, храниться, отыскиваться, совместно использоваться, передаваться, анализироваться и визуализироваться.

Анализ структурированных, неструктурированных или полуструктурированных данных помогает нам обнаруживать скрытые закономерности, тенденции рынка, корреляции, личные предпочтения и т. д. С помощью правильных инструментов обработки и анализа организация данных может привести к гораздо лучшим маркетинговым планам, дополнительным возможностям получения дохода, улучшению обслуживания клиентов, более здоровой операционной эффективности, конкурентным преимуществам и многому другому.

Каждая компания собирает данные и их использует; однако, чтобы обеспечить потенциальный успех, компания должна использовать данные эффективнее. Каждая компания должна создавать прямые ссылки на выведенные данные, которые могут улучшить бизнес прямо или косвенно.

Хорошо, теперь у вас есть большие данные, под которыми обычно подразумевается крупный объем данных, и вы выполняете анализ – разве это все, что вам нужно? Держитесь! Другим наиболее важным фактором является успешная монетизация данных. Итак, приготовьтесь и пристегните ремни – сейчас мы объясним важность больших данных!

В этой главе мы рассмотрим приведенные ниже разделы, чтобы выяснить роль больших данных в сегодняшней жизни и основные шаги установки реляционной СУБД MySQL 8:

- важность больших данных;
- жизненный цикл больших данных;

- что такое структурированная база данных;
- основы MySQL;
- новые функциональные средства, введенные в MySQL 8;
- преимущества использования MySQL 8;
- как установить MySQL 8;
- эволюция MySQL для больших данных.

Важность больших данных

Важность больших данных проистекает не только из того, сколько данных у вас есть, а, скорее, из того, что именно вы собираетесь с данными делать. Данные могут быть получены и проанализированы из непредсказуемых источников и могут быть использованы для решения многих вопросов. Давайте посмотрим на примеры использования, представляющие важность для реальной жизни, построенные на известных сценариях с помощью больших данных.

Следующий ниже рисунок помогает нам понять характер решения по обработке больших данных, обслуживающего различные отрасли промышленности. Хотя это не обширный список отраслей, в которых большие данные играют важную роль в бизнес-решениях, давайте обсудим эти несколько отраслей:

Социальные медиа

Контент социальных медиа – это информация, равно как и взаимодействия, такие как просмотры, лайки, демография, акции, читатели, уникальные посетите-

ли, комментарии и скачивания. Поэтому если рассматривать социальные медиа и большие данные, то они взаимосвязаны. В конце концов, важно то, как ваши усилия, связанные с социальными медиа, способствуют бизнесу.

Я наткнулся на одно замечательное высказывание: нет такого понятия, как доход от инвестиций в социальные медиа, – это во всех смыслах доход от инвестиций в бизнес.

Одним из примечательных примеров возможностей больших данных в Facebook является предоставление информации об образе жизни потребителей, шаблонах поиска, симпатиях, демографии, покупательских привычках и т. д. Facebook хранит около 100 Пб данных и накапливает 500 Тб данных почти ежедневно. С учетом количества подписчиков и собранных данных ожидается, что в ближайшие три года этот объем составит более 60 зеттабайт. Чем больше данных у вас есть, тем больше аналитической обработки вы можете проводить с помощью сложных прецизионных подходов для получения лучшей **отдачи от инвестиций** (ROI). Информация, полученная из социальных медиа, также используется при отслеживании целевой аудитории на предмет привлекательности и прибыльности рекламы.

Facebook имеет умный сервис под названием **Graph Search**, который помогает вам выполнять расширенный поиск по нескольким критериям. Например, вы можете искать *людей мужского пола, живущих в Ахмадабаде, которые работают в KNOWARTH Technologies*. Google также помогает вам уточнять поиск. Такие виды поиска и фильтров ими не ограничиваются; поисковый запрос также может содержать образование, политические взгляды, возраст и имя. Таким же образом вы еще можете запрашивать отели, фотографии, песни и многое другое. Вот здесь как раз и находится отдача от бизнес-инвестиций компании Facebook, которая предоставляет свои рекламные услуги, опирающиеся на конкретные критерии, такие как регионы, интересы или другие специфические особенности пользовательских данных. Google также предоставляет аналогичную платформу под названием **Google AdWords**.

Политика

Эра больших данных играет важную роль в политике; политические партии используют различные источники данных для отслеживания своих избирателей и улучшения своих избирательных кампаний. Аналитическая обработка больших данных также внесла значительный вклад в переизбрание Барака Обамы в 2012 году благодаря повышению вовлеченности и освещению именно тех тем, которые были важны для избирателей.

Нарендра Моди считается одним из самых технологичных и социально-медийно подкованных политиков в мире! У него почти 500 миллионов просмотров в Google+, 30 миллионов подписчиков в Twitter и 35 миллионов лайков в Facebook! Нарендра Моди состоит в партии **Бхартия Джанта** (BJP); анализ больших данных несет главную ответственность за успешные для партии BJP и ее партнеров Индийские всеобщие выборы в 2014 году с использованием инструментов с открытым исходным кодом, которые помогли им войти в прямой контакт со своими избирателями. BJP дотянулась до своих колеблющихся избирателей и даже до тех, кто не был настроен идти голосовать, поскольку они проводили мониторинг разговоров

в социальных сетях, отправляли соответствующие сообщения и использовали тактические приемы, чтобы улучшить свое видение избирательной кампании.

За семь месяцев загодя Нарендра Моди сделал заявление о приоритете туалетов перед храмами, после чего цифровая команда внимательно следила за разговорами в социальных сетях вокруг этого заявления. Было отмечено, что, по крайней мере, 50% пользователей были согласны с этим заявлением. Это был именно тот случай, когда возможность завоевать сердца избирателей была преобразована в миссию Swacch Bharat, что означает гигиеническую Индию. Результаты были ошеломляющими; поддержка партии BJP выросла почти до 30% всего за 50 часов.

Наука и исследование

Знаете ли вы, что с помощью больших данных расшифровка генома человека, которая на самом деле заняла 10 лет, теперь происходит едва ли не за день, а это почти в 100 раз меньше стоимости, предсказываемой законом Мура? Еще в 2000 году, когда **Sloan Digital Sky Survey** (SDSS, Слоановский цифровой небесный обзор), проект широкомасштабного исследования многоспектральных изображений и спектров красного смещения звезд и галактик, начал собирать астрономические данные, этот сбор происходил со скоростью около 200 Гб за ночь, что в то время было намного выше, чем данные, собранные за всю историю астрономии.

Национальное управление по аэронавтике и исследованию космического пространства (NASA) широко использует большие данные, учитывая огромный объем научных исследований. NASA собирает данные со всей Солнечной системы, чтобы раскрыть неизвестную информацию о Вселенной; его массивная коллекция данных является важным активом для науки и исследований и принесла пользу человечеству по различным направлениям. NASA получает данные, хранит их и эффективно использует самыми разными способами. Случаев использования данных агентства NASA так много, что их было бы трудно здесь перечислить!

Энергетика

Одна из ведущих энергетических компаний в Индии помогает улучшать потребление энергии с помощью предсказательного анализа больших данных, помогающего строить более прочные отношения с клиентами. Эта компания подключается к более чем 150 коммунальным услугам и обслуживает более 35 миллионов домашних хозяйств для улучшения использования энергии и снижения затрат и выбросов углерода. Она также предоставляет аналитические отчеты поставщикам коммунальных услуг, состоящих из более чем 10 миллионов точек данных каждый день, с целью целостного обзора использования для аналитических целей. Бытовые клиенты получают эти отчеты в счетах, которые показывают, где потребление энергии может быть уменьшено, и тем самым непосредственно помогают потребителям оптимизировать затраты на энергию.

Обнаружение мошенничества

Когда дело доходит до безопасности, обнаружения мошенничества или соблюдения законодательных требований, большие данные – это ваша вторая половинка, и именно если ваша вторая половинка помогает вам в выявлении и предотвращении проблем, прежде чем они ударят, то это становится «золотой серединой» для

бизнеса. Большую часть времени мошенничество обнаруживается уже после того, как оно произошло, когда вам, возможно, уже был нанесен ущерб. Последующие шаги, очевидно, будут минимизировать воздействие и улучшать области, которые могут помочь вам предотвратить его повторение.

Многие компании, которые вовлечены в самые разные типы обработки транзакций или претензий, широко используют методы обнаружения мошенничества. Платформы больших данных помогают им анализировать транзакции, претензии и т. д. в режиме реального времени, а также тенденции или аномальное поведение для предотвращения мошеннических действий.

Агентство национальной безопасности (АНБ) также занимается аналитической обработкой больших данных, чтобы срывать планы террористов. С помощью передовых методов обнаружения мошенничества на основе больших данных многие органы безопасности используют инструменты больших данных для прогнозирования преступной деятельности, мошенничества с кредитными картами, отлавливают преступников и предотвращают кибератаки. С каждым днем, по мере того как изменяются схемы мошенничества, несоблюдения законодательных требований и нарушения систем безопасности, соответственно, становятся все богаче методы противоборства мошенническим транзакциям со стороны органов безопасности, чтобы идти на шаг впереди для таких нежелательных сценариев.

Здравоохранение

В настоящее время трекер здоровья на запястье – это весьма обыденная вещь; однако с помощью больших данных он не только показывает вашу личную панель мониторинга или изменения показаний с течением времени, но и дает вам соответствующие предложения, основанные на медицинских данных, которые он собирает для улучшения вашего рациона, а также аналитические факты о таких людях, как вы. Таким образом, из простых наручных трекеров здоровья можно получить много сведений, которые могут улучшить здоровье пациента. Компании, предоставляющие такие услуги, также анализируют влияние на здоровье, прослеживая тенденции. Постепенно такие носимые устройства также начинают использоваться в отделениях неотложной медицинской помощи для быстрого анализа характера неотложных восстановительных мер со стороны врача.

Используя данные, накопленные государственными учреждениями, файлы социальных служб, отчеты о травматизме и клинические данные, больницы могут помочь оценить потребности в здравоохранении. Географические статистические данные, основанные на многочисленных факторах, от роста численности населения и заболеваемости до повышения качества жизни людей, сопоставляются для определения наличия медицинских услуг, машин скорой помощи, служб экстренной помощи, планов борьбы с пандемией и других соответствующих служб здравоохранения. Все эти меры, которые осуществляются несколькими учреждениями на регулярной основе для прогнозирования эпидемий гриппа, помогают свести к нулю вероятные экологические опасности, риски для здоровья и негативные тенденции.

Бизнес-картирование

Компания Netflix имеет миллионы подписчиков; она использует большие данные и аналитические данные о привычках подписчиков на основе возраста, пола и географического положения, чтобы индивидуально настраивать те аспекты, ко-

торые, как оказалось, генерируют большую деловую активность в соответствии с ожиданиями компании.

Компания Amazon еще в 2011 году начала присуждать $5 своим клиентам, которые используют мобильное приложение Amazon Price Check для сканирования продуктов в магазине, захвата изображения и поиска самых низких цен. Оно также имело возможность проставлять внутреннюю цену магазина на товары. Роль больших данных заключалась в том, чтобы всю информацию о товарах можно было увязать с товарами Amazon для сравнения цен и тенденций клиентов и, соответственно, планировать маркетинговые кампании и предложения на основе ценных данных, которые аккумулировались, чтобы доминировать на быстро развивающемся конкурентном рынке электронной коммерции.

Компания Макдональдс имеет более чем 35 000 местных ресторанов, которые обслуживают около 75 миллионов клиентов в более чем 120 странах. Она использует большие данные, чтобы получать представление о том, как улучшить опыт клиентов, и предлагает ключевые факторы компании, такие как состав меню, продолжительность ожидания в очереди, размер заказа и модель заказов клиентов, что в совокупности помогает им оптимизировать эффективность своих операций и индивидуальных настроек на основе географических местоположений для обеспечения прибыльности бизнеса.

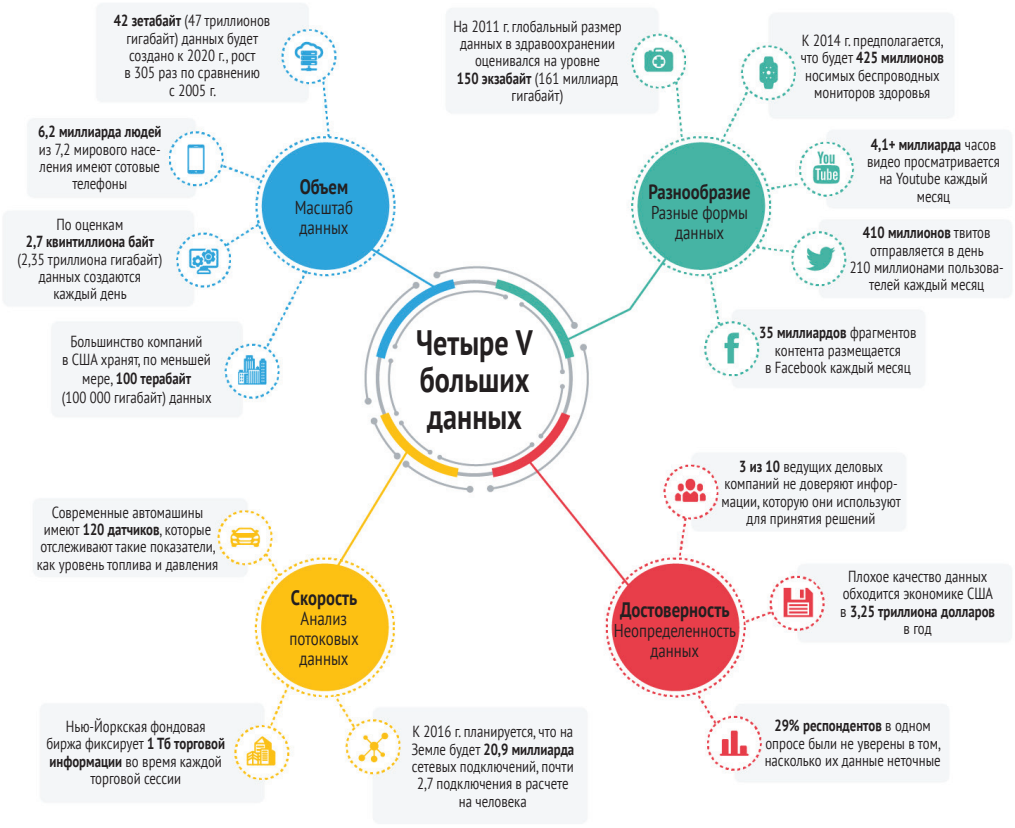
Есть много реальных случаев использования больших данных, которые изменили человечество, технологии, предсказания, здоровье, науку и исследования, закон и порядок, спорт, опыт клиентов, энергетику, финансовую торговлю, робототехнику и многие другие области. Большие данные – это неотъемлемая часть нашей повседневной жизни, которая не всегда очевидна, но, без всяких сомнений, во многих отношениях играет значительную роль в том, что мы делаем. Пришло время подробно рассмотреть то, как структурирован жизненный цикл больших данных. Это даст более четкую картину многих областей, которые играют значительную роль в размещении данных в том месте, где они могут быть использованы для обработки.

Жизненный цикл больших данных

Многие организации рассматривают большие данные не только как модное слово, но и как интеллектуальную систему для улучшения бизнеса и получения соответствующей информации и идей. Большие данные – это термин, который относится к управлению огромным объемом сложных необработанных данных из различных источников, таких как базы данных, социальные медиа, изображения, сенсорное оборудование, файлы журналов регистрации событий, мнения людей и т. д. Эти данные могут быть структурированными, полуструктурированными или неструктурированными. Поэтому для обработки таких данных, которая в условиях традиционных процедур обработки представляла бы собой сложный и трудоемкий процесс, используются специальные инструменты больших данных.

Жизненный цикл больших данных может быть сегментирован на **объем**, **разнообразие**, **скорость** и **достоверность** – обычно известные как **ЧЕТЫРЕ V** (Volume, Variety, Velocity и Veracity) **БОЛЬШИХ ДАННЫХ**. Давайте кратко их рассмотрим, а затем перейдем к четырем фазам жизненного цикла больших данных, то есть сбору, хранению, анализу и управлению данными.

Ниже показано несколько реальных сценариев, которые дают нам гораздо лучшее понимание четырех V и определения больших данных:



Объем

Объем подразумевает огромное количество данных, генерируемых и хранимых каждую секунду. Счет размера данных на предприятиях ведется уже не в терабайтах – он наращивается в зеттабайтах или бронтобайтах. Новые инструменты для работы с большими данными в настоящее время, как правило, используют распределенные системы, которые иногда могут быть диверсифицированы по всему миру.

Ожидается, что объем данных, полученных по всему миру к 2008 году, будет к 2020 году генерироваться всего за минуту.

Разнообразие

Разнообразие относится к нескольким типам и характеру данных, таких как потоки нажатий на веб-страницах, текст, датчики, изображения, голос, видео, файлы журналов регистрации событий, беседы в социальных сетях и многое другое. Это помогает людям, которые тщательно их изучают, эффективно их использовать для углубленного понимания.

70% данных в мире не структурированы, в частности текст, изображения, голос и т. д. Однако ранее структурированные данные были популярны из-за их доступности для анализа, поскольку они могут храниться в файлах, базах данных или поддаются традиционным процедурам хранения данных.

Скорость

Скорость подразумевает скорость генерирования, усвоения и обработки данных для удовлетворения потребностей и решения задач, которые возникают на пути эволюции и расширения.

Каналы связи нового поколения, такие как социальные сети, электронные письма и мобильные телефоны, прибавили скорости данным в больших данных. Процесс ежедневного отслеживания около 1 Тб информации о торговых событиях для выявления мошенничества чувствителен ко времени, когда иногда каждая минута имеет важное значение для предотвращения мошенничества. Просто представьте разговоры в социальных сетях, которые в считанные секунды могут становиться вирусными; на таких платформах анализ помогает нам выявлять тенденции.

Правдивость

Под достоверностью понимается несогласованность данных, которую можно обнаружить; она может повлиять на эффективное управление и обработку данных. Управление такими данными и придание им ценности – вот где могут помочь большие данные.

Когда мы говорим о больших данных, качество и точность остаются главной задачей. Разве не ради этого все крутится? Количество Twitter-каналов является подходящим вариантом применения, где изобилуют хештеги, опечатки, неофициальный текст и аббревиатуры; вместе с тем мы ежедневно сталкиваемся со сценариями, где большие данные отлично справляются со своей работой в серверной части и позволяют нам работать с таким типом данных.

Фазы жизненного цикла больших данных

Эффективное использование больших данных с экспоненциальным ростом в типах и объемах данных имеет огромный потенциал для преобразования экономической, деловой и маркетинговой информации и наращивания клиентской базы. Большие данные стали ключевой мантрой успеха для текущих конкурентных рынков, для существующих компаний и фактором, меняющим правила игры в конкурентной борьбе для новых компаний. Все это может оказаться истиной, только если задействуется **ЦЕННОСТЬ ДАННЫХ**. Давайте посмотрим на следующий ниже рисунок:

Как показано на рисунке, жизненный цикл больших данных можно разделить на четыре этапа. Рассмотрим их подробнее.

Сбор

Этот раздел является ключевым в жизненном цикле больших данных. Он определяет, какой тип данных фиксируется в источнике. В качестве примеров можно привести сбор журналов регистрации событий с сервера, извлечение профилей пользователей, автоматический обзор организаций для анализа мнений и сведений о заказах. Примеры, которые мы упомянули, могут включать в себя работу с локальным языком, текстом, неструктурированными данными и изображениями, в которых мы будем заинтересованы по мере продвижения вперед в жизненном цикле больших данных.

С повышением уровня автоматизации потоков коллекций данных меняются и организации, которые классически тратят много усилий на сбор структурированных данных для анализа и оценки ключевых точек успеха бизнеса. Зрелые организации теперь используют данные, в обычном случае игнорируемые из-за их размера или формата, которые в терминологии больших данных часто называются неструктурированными данными. Эти организации всегда стараются использовать максимальный объем информации, будь то структурированная или неструктурированная, так как для них данные представляют ценность как таковую.

Данные можно передать в платформу больших данных, такую как **HDFS** (Hadoop Distributed File System), и их там консолидировать. После того как данные обработаны с помощью таких инструментов, как Apache Spark, их можно загрузить обратно в базу данных MySQL, которая поможет заполнить соответствующими данными, чтобы показать, из каких составляющих MySQL состоит.

С ростом объемов данных и увеличением скорости теперь Oracle имеет интерфейс NoSQL, предназначенный для подсистем хранения данных InnoDB и MySQL Cluster. Подсистема MySQL Cluster дополнительно полностью обходит слой SQL. Без синтаксического анализа и оптимизации SQL данные в формате ключ-значение могут вставляться прямо в таблицы MySQL в девять раз быстрее.

Хранение

В этом разделе мы обсудим хранение данных, собранных из различных источников. Рассмотрим пример автоматических обзоров организаций для анализа мнений, где каждый обзор собирает данные с разных сайтов и по каждому отображаются уникальные данные.

Традиционно данные обрабатывались с помощью процедуры **ETL** (извлечение, трансформация и загрузка), которая использовалась для сбора данных из различных источников, их изменения в соответствии с требованиями и загрузки в хранилище для дальнейшей обработки или отображения. Для подобных сценариев очень часто использовались такие инструменты, как электронные таблицы, реляционные СУБД, инструменты бизнес-аналитики и т. д., а иногда и вручную.

Наиболее распространенным хранилищем, используемым в платформе больших данных, является HDFS. HDFS также предоставляет в распоряжение язык запросов **HQL** (Hive Query Language), который помогает выполнять многие аналитические задачи, которые традиционно выполняются инструментами бизнес-аналитики. Можно рассмотреть несколько других вариантов хранения – это Apache Spark, Redis и MongoDB. Каждый вариант хранения имеет свой собственный способ работы в серверной части; однако большинство поставщиков хранения предоставляет прикладные программные интерфейсы SQL, которые могут использоваться для дальнейшего анализа данных.

Также может возникнуть ситуация, когда нам нужно собирать и демонстрировать данные в реальном времени, которая практически не требует хранения данных для будущих целей и позволяет выполнять аналитическую обработку в реальном времени для получения результатов на основе запросов.

Анализ

В этом разделе мы обсудим, как эти различные типы данных анализируются на основе универсального вопроса, который начинается со слов «что, если…?». Эволюция организаций вместе с данными также повлияла на новые стандарты метаданных, организующие их с целью первичного обнаружения и переработки для структурных подходов, вызревающих на основе ценности создаваемых данных.

Большинство зрелых организаций надежно обеспечивают доступность, превосходство и ценность для бизнес-подразделений с постоянным автоматизированным процессом структурирования метаданных и результатов, которые будут обрабатываться для анализа. В зрелой организации, управляемой данными, механизм анализа обычно работает с несколькими источниками данных и типами данных, которые также включают в себя данные, поступающие в режиме реального времени.

В фазе анализа обрабатываются сырые данные, для которых СУБД MySQL имеет задания MapReduce в Hadoop, которые проводят анализ и выводят результат. Когда данные MySQL расположены в HDFS, к ним в целях дальнейшего анализа может обращаться остальная часть экосистемы инструментов, связанных с платформой больших данных.

Управление

Невозможно извлечь ценность данных для бизнеса без сформулированной политики управления данными на практике. В отсутствии продуманной политики управления данными предприятия могут столкнуться с неправильной интерпретацией информации, что в конечном итоге может привести к непредсказуемому ущербу для бизнеса. С помощью управления большими данными организация может достигнуть последовательной, точной и действенной осведомленности в данных.

Управление данными осуществляется с целью соблюдения законодательных требований, конфиденциальности, нормативно-юридических актов и всего, что является обязательным в соответствии с требованиями бизнеса. В целях управления данными непрерывный мониторинг, изучение, пересмотр и оптимизация качества процесса также должны учитывать потребности в безопасности данных. До сих пор, когда речь шла о больших данных, управление данными принималось с легкостью; однако, с ростом объемов данных и их повсеместным использованием управление данными стало привлекать к себе все больше внимания. Оно постепенно становится обязательным фактором для любого проекта в области больших данных.

Поскольку у нас есть хорошее понимание жизненного цикла больших данных, давайте теперь подробнее рассмотрим основы MySQL, ее преимущества и несколько превосходных функциональных средств этой реляционной СУБД.

Структурированные базы данных

Многие организации используют структурированную базу данных для хранения своих данных в организованном виде в отформатированном хранилище. В сущности, данные в структурированной базе данных имеют фиксированное поле, предопределенную длину данных, и в них установлено, какие данные там должны храниться, в частности числа, дата, время, адрес, валюта и т. д. Короче говоря, структура уже определена до того, как данные будут вставляться, что дает более четкое представление о том, какие данные могут там находиться. Ключевым преимуществом использования структурированной базы данных является простота хранения, запросов и анализа данных.

Неструктурированная база данных – это полная противоположность; она не имеет идентифицируемой внутренней структуры. Она может иметь массивный неорганизованный агломерат или различные объекты. Источник структурированных данных главным образом генерируется машиной, имея в виду, что информация поступает от машины и без вмешательства человека, в то время как неструктурированные данные генерируются человеком. Организации используют структурированные базы данных для таких данных, как транзакции банкоматов, бронирование авиабилетов, системы инвентаризации и т. д. Точно так же некоторые организации используют неструктурированные данные, такие как электронные письма, мультимедийный контент, текстовые документы, веб-страницы, бизнес-документы и т. д.

Структурированные базы данных – это традиционные базы данных, которые используются многими предприятиями уже более 40 лет. Тем не менее в современном мире объем данных становится все больше и больше, и возникла общая

потребность – аналитическая обработка данных. Со структурированными базами данных аналитика становится все сложнее, поскольку объем и скорость цифровых данных растут быстрее с каждым днем; нам нужно найти способ удовлетворения таких потребностей эффективным и действенным образом. Наиболее распространенной в мире СУБД, которая используется в качестве структурированной базы данных с открытым исходным кодом, является MySQL. Вы познакомитесь с тем, как сделать эту структурированную СУБД пригодной для обработки больших данных, что в итоге приведет к упрощению комплексного анализа. Прежде всего давайте в следующем разделе рассмотрим некоторые идеи MySQL.

Основы MySQL

MySQL – это структурированная реляционная система управления базами данных с открытым исходным кодом, хорошо известная в силу ее производительности, простоты в использовании и надежности. Это наиболее распространенный вариант для веб-приложений на основе реляционной базы данных. На текущем рынке тысячи веб-приложений опираются на MySQL, включая такие гиганты отрасли, как Facebook, Twitter и Wikipedia. Она также зарекомендовала себя в качестве хорошего варианта для **SaaS** (программное обеспечение как служба) на основе таких приложений, как SugarCRM, Supply Dynamics, Workday, RightNow, Omniture и Zimbra. MySQL была разработана шведской компанией MySQL AB, и теперь она распространяется и поддерживается корпорацией Oracle.

MySQL как реляционная система управления базами данных

Данные в реляционной базе данных хранятся в организованном формате, который позволяет легко извлекать информацию. Данные хранятся в различных таблицах, состоящих из строк и столбцов. Вместе с тем также может быть настроена связь между различными таблицами, в которых эффективно хранятся огромные объемы данных и из которых эффективно извлекаются отобранные данные. Это позволяет выполнять операции базы данных с огромной скоростью и гибкостью.

Как реляционная СУБД, MySQL имеет возможности устанавливать связи с различными таблицами по схеме один-ко-многим, многие-к-одному и один-к-одному, предоставляя первичные ключи, внешние ключи и индексы. Для получения точной информации мы также можем выполнять соединения между таблицами, такие как внутренние соединения и внешние соединения.

В MySQL для взаимодействия с реляционными данными в качестве интерфейса используется **язык структурированных запросов SQL** (Structured Query Language). SQL является стандартным, согласно **ANSI** (Американскому национальному институту стандартов), языком, с помощью которого мы можем оперировать данными, выполняя такие операции, как создание, удаление, обновление и извлечение.

Лицензирование

Многие отрасли предпочитают технологии с открытым исходным кодом в силу их гибкости и экономии средств, тогда как MySQL оставила свой след на рынке, став самой популярной реляционной СУБД для веб-приложений. Открытый исходный код означает, что вы можете просматривать исходный код MySQL и настраивать

его под свои потребности без каких-либо затрат. Вы можете скачать исходные или двоичные файлы с сайта MySQL и использовать их по своему усмотрению.

Сервер MySQL подпадает под действие лицензии **GNU** (General Public License, универсальная общедоступная лицензия), что означает, что мы можем свободно его использовать для веб-приложений, соответствующим образом изучать и изменять его исходный код. Он также имеет корпоративную версию с расширенными функциональными возможностями. Многие предприятия приобретают у MySQL корпоративную поддержку, чтобы получать помощь по различным вопросам.

Надежность и масштабируемость

СУБД MySQL работает очень надежно, не требуя широкомасштабного устранения проблем из-за узких мест или других замедлений. Она также включает в себя ряд улучшающих производительность механизмов, таких как поддержка индексов, утилиты загрузки и кеши памяти. MySQL использует InnoDB как подсистему хранения данных, которая обеспечивает очень эффективные ACID-совместимые (с поддержкой транзакционной семантики) транзакционные возможности, гарантирующие высокую производительность и масштабируемость. Для обработки быстро растущей базы данных масштабировать ее помогают подсистемы MySQL Replication и MySQL Cluster.

Совместимость платформ

СУБД MySQL имеет большую кросс-платформенную доступность, что делает ее популярнее. Она гибко работает на основных платформах, таких как Red Hat, Fedora, Ubuntu, Debian, Solaris, Microsoft Windows и Apple macOS. Она также предоставляет **прикладной программный интерфейс** (API) для взаимодействия с различными языками программирования, такими как C, C++, C#, PHP, Java, Ruby, Python и Perl.

Выпуски

Вот список главных версий MySQL, выпущенных до настоящего времени:

- версия 5.0 GA была выпущена 19 октября 2005 г.;
- версия 5.1 GA была выпущена 14 ноября 2008 г.;
- версия 5.5 GA была выпущена 3 декабря 2010 г.;
- версия 5.6 GA была выпущена 5 февраля 2013 г.;
- версия 5.7 GA была выпущена 21 октября 2015 г.

Теперь пришло время для выпуска основной версии – MySQL 8, которая была анонсирована 12 сентября 2016 г. и пока еще находится в режиме разработки. Давайте посмотрим, что нового появилось в последней версии.

Новые возможности в MySQL 8

Команда разработчиков СУБД MySQL недавно анонсировала свой основной релиз как MySQL 8 **Development Milestone Release** (DMR) со значительными обновлениями и исправлениями проблем, которые были очень необходимы в изменениях больших данных.

Вам может быть интересно, почему номер версии 8 сразу после 5.7! Разве не пропущены промежуточные версии, то есть номера 6 и 7? Конечно, нет; фактически номер 6.0 был сохранен в рамках перехода к более частому и своевременному выпуску, в то время как номер 7.0 оставлен для кластерной версии MySQL.

Давайте посмотрим на некоторые интересные функциональные возможности, представленные в этой последней версии, показанные на следующей ниже схеме:

Самое время подробно рассмотреть функциональные возможности MySQL 8, которые вдохновляют и убеждают в необходимости крупного обновления версии MySQL.

Транзакционный словарь данных

В предыдущей версии словарь данных СУБД MySQL хранился в разных файлах метаданных и нетранзакционных таблицах, но начиная с этой версии она будет иметь транзакционный словарь данных для хранения информации о базе данных. Больше нет файлов `.frm`, `.trg` или `.par`. Вся информация будет храниться в базе данных, что снимает затраты на выполнение тяжелых файловых операций. Были многочисленные проблемы с хранением метаданных файловой системы, такие как уязвимость файловой системы, непомерные файловые операции, которые

с трудом справляются с восстановлением после аварийного завершения работы или репликацией; было также трудно добавлять новые метаданные, связанные с каким-то функциональным средством. Теперь это обновление упрощает централизованное хранение информации и повышает производительность, поскольку этот объект-словарь данных может кешироваться в памяти, как и другие объекты базы данных.

Этот словарь данных будет содержать данные, необходимые для выполнения SQL-запроса, в частности каталожную информацию, наборы символов, параметры сортировки, типы столбцов, индексы, сведения о базе данных, таблицы, процедуры хранения, функции и триггеры и т. д.

Роли

В MySQL 8 модуль привилегий был улучшен путем введения ролей, то есть коллекции разрешений. Теперь мы можем создавать роли с несколькими привилегиями и назначать их нескольким пользователям.

Проблема предыдущей версии была в том, что мы не могли определять общие разрешения для группы пользователей и каждый пользователь имел индивидуальные привилегии. Предположим, что если уже существует 1000 пользователей с общими привилегиями и вы хотите удалить разрешения на запись для всех этих 1000 пользователей, то что бы вы сделали в предыдущей версии? Вы бы предприняли длительный подход к обновлению каждого пользователя, не так ли? Брр… трудоемкая задача.

Теперь с MySQL 8 любое изменение привилегий легко обновляется. Роли будут определять все необходимые привилегии, и эта роль будет назначена этим 1000 пользователям. Нам просто нужно внести какие-либо изменения привилегий в роли, и все пользователи будут автоматически наследовать соответствующие привилегии.

Роли можно создавать, удалять, предоставлять или отзывать разрешения, предоставлять или отзывать из учетной записи пользователя, а также указывать роль по умолчанию в текущем сеансе.

Автоинкремент InnoDB

СУБД MySQL 8 изменила механизм хранения значения автоинкрементного счетчика. Ранее он хранился в памяти, что приводило к значительным сложностям в управлении во время перезапуска или сбоев сервера. Однако теперь значение автоинкрементного счетчика записывается в журнал повтора при каждом изменении его значения, и на каждой контрольной точке оно сохраняется в системной таблице, что делает его непрерывным при перезапуске сервера.

В предыдущей версии обновление значения автоинкрементного счетчика могло вызвать ошибки дублирования при вводе. Предположим, если вы обновили значение автоинкремента в середине последовательности большим значением, чем текущее максимальное значение, тогда только последующие операции вставки смогут определить неиспользуемые значения, что может вызвать проблему повторяющейся записи. Это было предотвращено путем обеспечения непрерывности автоинкрементного значения, поэтому последующие операции вставки могут получить новое значение и правильно его назначить.

Если происходил перезапуск сервера, то в предыдущей версии автоинкрементное значение терялось, поскольку оно хранилось в памяти, и InnoDB должен был выполнить запрос, чтобы узнать максимальное используемое значение. Это было изменено, поскольку более новая версия имеет возможность поддерживать его значение непрерывным во всех перезапусках сервера. Во время перезапуска сервера InnoDB инициализирует значение счетчика в памяти, используя максимальное значение, сохраненное в таблице словаря данных. В случае сбоев сервера InnoDB инициализирует значение автоинкрементного счетчика, которое больше, чем в таблице словаря данных и журнале повтора/отката.

Поддержка невидимых индексов

СУБД MySQL 8 предоставляет возможность делать индексы невидимыми. Такие индексы не могут использоваться оптимизатором. В случае, если вы хотите проверить производительность запросов без индексов, используя это функциональное средство, это можно выполнить, сделав их невидимыми, а не отбрасывая и повторно добавляя индекс. Это довольно удобное функциональное средство, когда индексирование должно отбрасываться и воссоздаваться на огромных наборах данных.

По умолчанию все индексы являются видимыми. Чтобы сделать их невидимыми или видимыми, используются ключевые слова соответственно `INVISIBLE` и `VISIBLE`, как описано в следующем ниже фрагменте кода:

```
ALTER TABLE таблица1 ALTER INDEX ix_столбца1_таблицы1 INVISIBLE;
ALTER TABLE таблица1 ALTER INDEX ix_столбца1_таблицы1 VISIBLE;
```

Улучшение индексов, отсортированных по убыванию

Нисходящие индексы существовали и в версии 5.7, но они сканировались в обратном порядке, что создавало барьеры производительности. Чтобы улучшить производительность, СУБД MySQL 8 это оптимизирует и сканирует нисходящие индексы в прямом порядке, что резко улучшило производительность. Она также передает в оптимизатор многостолбцовые индексы, когда наиболее эффективный порядок сканирования для некоторых столбцов возрастающий и для других столбцов убывающий.

SET PERSIST

Серверные переменные могут быть настроены глобально и динамически во время работы сервера. Существует множество системных переменных, которые можно установить с помощью `SET GLOBAL`:

```
SET GLOBAL max_connections = 1000;
```

Однако такие настройки будут потеряны после перезапуска сервера. Чтобы избежать этого, MySQL 8 ввел параметр `SET PERSIST`, который сохраняет переменные во всех перезапусках сервера:

```
SET PERSIST max_connections = 1000;
```

Расширенная поддержка ГИС

В предыдущей версии СУБД поддерживала только одну систему координат, безразмерное двумерное положение, которое не было привязано к положению на

земле. Теперь СУБД MySQL 8 добавила поддержку для системы пространственной привязки **Spatial Reference System** (SRS) с геопривязанными эллипсоидами и двумерными проекциями. SRS помогает назначать местоположению координаты и устанавливает связи между наборами таких координат. Этими пространственными данными можно управлять в хранилище словаря данных в виде таблицы `ST_SPATIAL_REFERENCE_SYSTEMS`.

Кодировка символов по умолчанию

Кодировка символов по умолчанию была изменена с `latin1` на UTF8. UTF8 является доминирующим набором символов, хотя это не было по умолчанию в предыдущих версиях MySQL. Наряду с кодировкой символов по умолчанию, параметры сортировки были изменены с `latin1_swedish_ci` на `utf8mb4_800_ci_ai`. С этими глобально принятыми изменениями наборы символов и параметры сортировки теперь основаны на UTF8; одна из распространенных причин заключается в том, что UTF8 поддерживает около 21 различных языков, что позволяет системам обеспечивать многоязычную поддержку.

Улучшение побитовых операций

В MySQL 5.7 побитовые операции и функции работали только для типов данных `Bigint` (64-разрядных целых чисел). Нам нужно было передавать `BIGINT` в качестве аргумента, и она возвращала результат как `BIGINT`. Короче говоря, для выполнения операций он имел максимальный диапазон до 64 разрядов. Если пользователь хотел выполнить их для других типов данных, то ему нужно было выполнять преобразование в тип данных `BIGINT`. Такая типизация невозможна для типов данных, размер которых превышает 64 разряда, так как она приведет к усечению фактического значения и в конечном счете к неточности в данных.

В СУБД MySQL 8 усовершенствованы побитовые операции путем включения поддержки других двоичных типов данных, таких как `Binary`, `VarBinary` и `BLOB`. Это позволяет выполнять побитовые операции с большей, чем 64-разрядные данные, длиной. Типизация больше не требуется! Это позволяет принимать аргументы и возвращать результаты размером более 64 разрядов.

InnoDB Memcached

Множественные операции чтения сейчас возможны с плагином InnoDB Memcached, который действительно помогает в улучшении их производительности. Теперь множественные пары ключ-значение могут выбираться в одиночном запросе Memcached. Кроме того, был минимизирован частый коммуникационный трафик, так как мы можем получать многочисленные данные за один раз. Мы подробно рассмотрим плагин Memcached в главе 4 «*Использование Memcached с MySQL 8*».

Плагин InnoDB Memcached также поддерживает диапазонные запросы. Это упрощает поиск по диапазону, позволяя задавать определенный диапазон и извлекать значения в этом диапазоне.

NOWAIT и SKIP LOCKED

Когда строки блокируются другими транзакциями, к которым вы пытаетесь получить доступ, вам нужно дождаться, пока эта транзакция освободит блокировку

данной строки, чтобы вы могли получить к ней соответствующий доступ. Чтобы избежать ожидания другой транзакции, InnoDB добавил поддержку параметров NOWAIT и SKIP LOCKED. Параметр NOWAIT немедленно возвращается с ошибкой, в случае если запрошенная строка заблокирована, а не переходит в режим ожидания, а параметр SKIP LOCKED пропускает заблокированную строку и не ждет получения блокированной строки. Следовательно, SKIP LOCKED не будет учитывать блокированную строку в результирующем наборе:

```
SELECT * FROM таблица1 WHERE id = 5 FOR UPDATE NOWAIT;
SELECT * FROM таблица1 FOR UPDATE SKIP LOCKED;
```

Преимущества использования MySQL

Независимо от того, являетесь ли вы разработчиком или представляете предприятие, вы, очевидно, выберете технологию, которая обеспечивает хорошие преимущества и результаты по сравнению с другими аналогичными продуктами. СУБД MySQL предоставляет многочисленные преимущества в качестве наилучшего выбора на этом конкурентном рынке. Она имеет различные мощные функциональные средства, которые делают ее более всеобъемлющей СУБД. Давайте теперь рассмотрим некоторые преимущества использования MySQL.

Безопасность

Первое, что приходит на ум, – это обеспечение безопасности данных, потому что в настоящее время данные стали ценными и вполне могут повлиять на непрерывность деятельности предприятия, если юридические обязательства не выполняются; на самом деле все может быть настолько плохо, что ваше предприятие может закрыться в самые кратчайшие сроки. MySQL является наиболее безопасной и надежной системой управления базами данных, используемой многими известными предприятиями, такими как Facebook, Twitter и Wikipedia. Она действительно обеспечивает хороший уровень безопасности, который защищает конфиденциальную информацию от злоумышленников. MySQL обеспечивает управление контролем доступа так, чтобы предоставление и отзыв необходимого доступа у пользователя было легким. Кроме того, могут быть определены роли со списком разрешений, которые пользователю могут предоставляться или отменяться. Все пароли пользователей хранятся в зашифрованном формате с использованием специфичных для плагина алгоритмов.

Масштабируемость

С каждым днем растет гора данных вследствие широкого применения технологий по самым разным направлениям. За счет этого средняя нагрузка взлетает до небес. В некоторых случаях уже совершенно невозможно предсказать, превысят ли данные некоторый предел или не выйдет ли количество пользователей за пределы. Масштабируемые базы данных были бы предпочтительным решением, чтобы в любой момент мы могли удовлетворить неожиданные требования к масштабированию. MySQL – это система управления базами данных, наилучшим образом приспособленная для масштабируемости, которая может масштабироваться горизонтально и вертикально; с точки зрения данных и нагрузки, вызываемой при-

кладными запросами в многочисленных серверах, MySQL вполне предсказуема. Для обработки возросшей нагрузки в MySQL Cluster легко добавляется дополнительная вычислительная мощность.

Реляционная система управления базами данных с открытым исходным кодом

MySQL – это реляционная система управления базами данных с открытым исходным кодом, которая делает отладку, обновление и расширение функциональности быстрым и легким. Вы можете просмотреть источник, внести соответствующие изменения и использовать его по своему усмотрению. Вы также имеете возможность распространять расширенную версию MySQL, но для этого потребуется лицензия.

Высокая производительность

СУБД MySQL обеспечивает обработку высокоскоростных транзакций с оптимальной скоростью. Она может кешировать результаты, что повышает производительность чтения. Репликация и кластеризация позволяют улучшать параллелизм и управлять рабочей нагрузкой. Индексы базы данных также ускоряют выполнение инструкций с запросом `SELECT` для больших объемов данных. Чтобы повысить производительность, в схему производительности СУБД MySQL 8 были включены индексы для ускорения извлечения данных.

Высокая доступность

В сегодняшнем мире конкурентного маркетинга ключевым моментом организации является поддержка системы в рабочем состоянии. Любые аварийные завершения работы или простои в нерабочем состоянии непосредственно влияют на предпринимательскую деятельность и доход; следовательно, высокая доступность является фактором, который нельзя упускать из виду. СУБД MySQL довольно надежна и имеет постоянную доступность благодаря кластерной и репликационной конфигурациям. Кластерные серверы мгновенно обрабатывают аварийное завершение работы и управляют аварийным переключением, чтобы поддерживать доступность системы почти постоянно. Если один сервер падает, он перенаправит запрос пользователя на другой узел и выполнит запрошенную операцию.

Кросс-платформенность

СУБД MySQL обеспечивает кросс-платформенную гибкость, которая может работать на различных платформах, таких как Windows, Linux, Solaris, OS 2 и т. д. Она имеет отличную поддержку API для всех основных языков, что позволяет ее очень легко интегрировать с такими языками, как PHP, C++, Perl, Python, Java и т. д. Она также является частью сервера LAMP (Linux Apache MySQL PHP), который используется во всем мире для веб-приложений.

Теперь пришло время закатать рукава и взглянуть на MySQL 8; давайте начнем с установки MySQL 8, в нашем случае на платформе Linux. Мы предпочитаем размещать MySQL 8 в операционной системе Linux, поскольку это наиболее широко применяемый случай использования во многих организациях. Вы можете использовать ее на других платформах, которые поддерживаются MySQL, таких

как Windows, Solaris, HP – UNIX и т. д. Linux предоставляет различные способы установки сервера MySQL, такие как:

- пакет RPM;
- репозиторий YUM;
- репозиторий APT;
- репозиторий SLES;
- пакет Debian;
- пакет TAR;
- компиляция и инсталляция из исходного кода.

Мы установим MySQL 8 с помощью Linux-дистрибутива на основе RPM, предоставленного Oracle; однако вы можете выбрать любой из упомянутых здесь подходов. Давайте посмотрим, как получить и установить его с помощью пакета RPM.

Инсталляция MySQL 8

Давайте сначала посмотрим, откуда скачать MySQL 8, и разберемся в структуре пакетов, чтобы выбрать, какой из них подходит. Мы начинаем с получения MySQL 8, а затем быстро пройдемся по установке и верификации.

Получение MySQL 8

Скачайте пакет RPM сервера MySQL Community Server со страницы скачивания (https://dev.mysql.com/downloads/mysql/8.0.html). В зависимости от операционной системы и ее архитектурной версии имеются различные варианты. MySQL Community Server поставляется различными пакетами, которые могут быть описаны по имени пакета. Следующая ниже синтаксическая конструкция описывает пакет:

```
имяПакета-версия-дистрибутив-типАрхитектуры.rpm
```

Имя пакета: название пакета, в частности `myql-community-server`, `mysql-communityclient`, `mysql-community-libs`.

Версия: описывает версию конкретного пакета.

Дистрибутив: говорит о том, что пакет предназначен для дистрибутива Linux, основанного на его аббревиатуре.

Аббревиатура	Дистрибутив Linux
el6, el7	Red Hat Enterprise Linux/Oracle Linux/CentOS 6, 7
fc24, fc25	Fedora 24 или 25
sles12	SUSE Linux Enterprise Server 12
solaris11	Oracle Solaris 11

Тип архитектуры: описывает тип процессора, для которого был создан пакет, например x86_64, i686 и т. д.

Инсталляция MySQL 8

После того как у вас есть пакет RPM, просто установите его с помощью следующей ниже команды. Это позволит разместить необходимые файлы и папки в системных каталогах:

```
rpm -vih <имя-пакета>.rpm
```

Стандартная установка требует только пакетов `mysql-community-common`, `mysql-community-libs`, `mysql-community-client`, и `mysql-community-server`. Посмотрите на следующий ниже снимок экрана с процессом инсталляции:

```
[root@ip-172-31-43-204 tmp]# rpm -ivh mysql-community-common-8.0.1-0.1.dmr.el6.x86_64.rpm
warning: mysql-community-common-8.0.1-0.1.dmr.el6.x86_64.rpm: Header V3 DSA/SHA1 Signature, key ID 5072e1f5: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
   1:mysql-community-common-8.0.1-0.1.################################# [100%]
[root@ip-172-31-43-204 tmp]# rpm -ivh mysql-community-libs-8.0.1-0.1.dmr.el6.x86_64.rpm
warning: mysql-community-libs-8.0.1-0.1.dmr.el6.x86_64.rpm: Header V3 DSA/SHA1 Signature, key ID 5072e1f5: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
   1:mysql-community-libs-8.0.1-0.1.dm################################# [100%]
[root@ip-172-31-43-204 tmp]# rpm -ivh mysql-community-client-8.0.1-0.1.dmr.el6.x86_64.rpm
warning: mysql-community-client-8.0.1-0.1.dmr.el6.x86_64.rpm: Header V3 DSA/SHA1 Signature, key ID 5072e1f5: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
   1:mysql-community-client-8.0.1-0.1.################################# [100%]
[root@ip-172-31-43-204 tmp]# rpm -vih mysql-community-server-8.0.1-0.1.dmr.el6.x86_64.rpm
warning: mysql-community-server-8.0.1-0.1.dmr.el6.x86_64.rpm: Header V3 DSA/SHA1 Signature, key ID 5072e1f5: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
   1:mysql-community-server-8.0.1-0.1.################################# [100%]
```

После успешной инсталляции проверьте версию пакета, чтобы убедиться, что он установлен правильно и может быть доступен:

```
[root@ip-172-31-43-204 tmp]# mysql --version
mysql  Ver 8.0.1-dmr for Linux on x86_64 (MySQL Community Server (GPL))
[root@ip-172-31-43-204 tmp]#
```

Следующий шаг – переустановить временный пароль во время постинсталляции. MySQL ограничит использование базы данных, если не изменить этот временный пароль. Временный пароль будет сгенерирован MySQL и доступен в его лог-файле. Чтобы получить временный пароль, необходимо открыть файл `/var/log/mysql/mysqld.log` и отыскать ключевое слово `temporary password`.

Скопируйте его и попытайтесь подключиться к MySQL с помощью следующей ниже команды:

```
[root@ip-172-31-43-204 tmp]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.1-dmr

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Как показано на предыдущем снимке экрана, вы окажетесь в командной строке MySQL, где для сброса пароля можно выполнить следующий ниже запрос:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY '<НОВЫЙ_ПАРОЛЬ>';
```

Служебные команды MySQL

Давайте посмотрим на несколько элементарных команд MySQL:

- чтобы запустить службу MySQL, выполните следующую ниже команду:

```
service mysqld start
```

- чтобы остановить службу MySQL, выполните следующую ниже команду:

```
service mysqld stop
```

- чтобы проверить состояние службы MySQL, запущена она или нет, выполните следующую ниже команду:

```
service mysqld status
```

Эволюция MySQL для больших данных

Большинство предприятий в течение многих десятилетий используют MySQL в качестве реляционной СУБД. В их распоряжении находится большое количество данных, используемых либо для транзакций, либо для анализа собираемых и генерируемых данных, и именно здесь как раз и должны быть внедрены аналитические инструменты больших данных. Это теперь стало возможным с интеграцией MySQL с Hadoop. С помощью Hadoop данные можно хранить в подсистеме распределенного хранения, и вы также имеете возможность реализовать кластер Hadoop для распределенной аналитической подсистемы с целью анализа больших данных. Вычислительная среда Hadoop является наиболее предпочтительной в силу ее массивной параллельной обработки и мощных вычислений. С объединением MySQL и Hadoop теперь стало возможным получать аналитику в режиме реального времени, где Hadoop может хранить данные и работать параллельно с MySQL, чтобы показывать конечные результаты в режиме реального времени; это помогает обратиться ко многим случаям использования наподобие информации ГИС, которая была объяснена в разделе «*Введение в MySQL 8*» этой главы. Ранее мы познакомились с жизненным циклом больших данных, где данные могут преобразовываться для получения аналитических результатов. Давайте посмотрим, как MySQL вписывается в этот жизненный цикл.

Следующая ниже схема иллюстрирует то, как MySQL 8 накладывается на каждый из четырех этапов жизненного цикла больших данных:

Получение данных в MySQL

С ростом объема и скорости данных становится трудно передавать данные в MySQL с оптимальной производительностью. Чтобы избежать этого, Oracle разработала API NoSQL для хранения данных в подсистеме хранения данных InnoDB. Она не делает никакого анализа и оптимизации SQL, следовательно, данные ключ-значение могут напрямую записываться в таблицы MySQL с высокоскоростными транзакционными откликами без ущерба для гарантий ACID-совместимости. Подсистема MySQL Cluster также поддерживает различные API NoSQL для Node.js, Java, JPA, HTTP/REST и C++. Все это мы рассмотрим подробно позже в этой книге, однако мы должны иметь в виду, что с помощью API NoSQL мы также можем активировать более быструю обработку данных и транзакций в MySQL.

Организация данных в Hadoop

Следующим шагом является организация данных в файловой системе Hadoop, после того как данные были получены и загружены в MySQL. Большие данные требуют некоторой обработки для получения результатов анализа, где Hadoop используется для выполнения высокопараллельной обработки. Hadoop также является масштабируемой распределенной платформой и мощным средством с точки зрения вычислений. Здесь данные консолидируются из различных источников для аналитической обработки. Для передачи данных между таблицами MySQL в HDFS будет использоваться инструмент Apache Sqoop.

Аналитическая обработка данных

Теперь пришло время для анализа данных! Это фаза, где данные MySQL будут обработаны, используя алгоритм Hadoop MapReduce. Чтобы получить аналогичные аналитические результаты, мы можем использовать другие инструменты анализа, такие как Apache Hive или Apache Pig. Мы также можем выполнить собственный анализ, который может быть реализован на Hadoop, возвратив набор результатов с проанализированными и обработанными данными.

Результаты анализа

Результаты, которые были проанализированы в наших предыдущих фазах, загружаются обратно в MySQL; это может быть сделано с помощью Apache Sqoop. Теперь MySQL имеет результат анализа, который может использоваться для инструментов бизнес-аналитики, таких как Oracle BI, Jasper Soft, Talend и т. д., или другими традиционными способами, используя веб-приложения, которые могут формировать различные аналитические отчеты и при необходимости выполнять различные виды обработки в режиме реального времени.

Именно так MySQL легко вписывается в решение по обработке больших данных. Эта архитектура позволяет структурированным базам данных выполнять анализ больших данных. Чтобы понять, как этого достичь, обратитесь к главе 9 «*Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop*» и к главе 10 «*Практический пример: часть II. Обработка событий в реальном времени с помощью MySQL Applier*», в которых рассматривается несколько решений реальных задач, где мы широко обсуждаем применения MySQL 8 и решения бизнес-задач, связанных с генерированием ценностей из данных.

Резюме

В этой главе мы рассмотрели большие данные и их значение в различных отраслях. Мы также рассмотрели различные сегменты больших данных и их фазы жизненного цикла, включая сбор, хранение, анализ и управление. Мы обсудили структурированные базы данных и почему для больших данных нам нужна структурированная база данных. Мы рассмотрели основные функциональные средства реляционной СУБД MySQL и изучили недавно добавленные в MySQL 8 функциональные средства. Затем мы обсудили основные преимущества использования MySQL и узнали, как установить MySQL 8 в системе Linux с помощью пакета RPM. Наконец, мы разобрались в том, как СУБД MySQL может использоваться в анализе больших данных и как она может вписываться в фазы жизненного цикла больших данных с помощью Hadoop и Apache Sqoop.

В следующей главе вы изучите различные методы запроса в MySQL 8, включая соединения и агрегирования данных.

Глава 2

Методы запроса данных в MySQL 8

В главе 1 «*Введение в большие данные и MySQL 8*» мы увидели обзор больших данных и MySQL 8 наряду с новыми функциональными средствами MySQL 8 с последующей инсталляцией MySQL 8. В этой главе мы рассмотрим методы запроса данных. Эта глава предполагает, что вы просмотрели предыдущую главу и знакомы с MySQL 8 и с тем, как устанавливать MySQL 8.

В этой главе мы рассмотрим следующие важные темы, связанные с методами запроса в MySQL 8:

- обзор SQL;
- подсистемы (движки) хранения и их разновидности;
- инструкции выборки данных в MySQL;
- инструкции вставки и удаления в SQL;
- транзакции в MySQL;
- агрегирование данных в MySQL 8;
- тип данных JSON, выбор и обновление в типе данных JSON.

Прежде чем мы перейдем к деталям, давайте рассмотрим SQL в применении к MySQL.

Обзор SQL

Язык структурированных запросов (SQL) используется для управления, извлечения, вставки, обновления и удаления данных в **реляционной системе управления базами данных** (реляционной СУБД). Чтобы сделать проще, SQL сообщает базе данных, что нужно сделать и что именно ей нужно. SQL является стандартным языком, который используется во всех СУБД, таких как MySQL, MS Access, MS SQL, Oracle, Postgres, и др.

Используя SQL, пользователь может получать доступ, определять и манипулировать данными из MySQL. Мы можем использовать SQL-модули, библиотеки и прекомпиляторы для встраивания в другие языки – все это помогает создавать/удалять базы данных и таблицы, управлять представлениями, хранимыми процедурами, функциями и т. д. в базе данных. С помощью SQL мы также можем управлять разрешениями для таблиц, процедурами и различными представлениями.

Ниже приведено несколько важных команд SQL, которые мы объясним более подробно в этой главе:

- **SELECT**: извлекает данные из базы данных;
- **UPDATE**: обновляет данные из базы данных;
- **DELETE**: удаляет существующие записи из базы данных;
- **INSERT INTO**: добавляет новую информацию в базу данных;
- **CREATE DATABASE**: создает новую базу данных;
- **ALTER DATABASE**: модифицирует или изменяет характеристики базы данных;
- **DROP DATABASE**: удаляет базу данных;
- **CREATE TABLE**: создает новую таблицу;
- **ALTER TABLE**: модифицирует или изменяет характеристики таблицы;
- **DROP TABLE**: удаляет таблицу;
- **CREATE INDEX**: создает индекс;
- **DROP INDEX**: удаляет индекс.

Подсистемы (движки) хранения и их разновидности

Давайте посмотрим на различные подсистемы хранения базы данных MySQL. Прежде чем перейти к методам запроса данных, очень важно разобраться с информацией этого раздела, поскольку подсистемы хранения данных играют важную роль в методах запроса данных. MySQL хранит данные в базе данных как подкаталог. В каждой базе данных данные хранятся в таблицах, а каждое определение таблицы хранится в файле с расширением `.frm` с тем же именем, что и имя таблицы. Предположим, что если мы создадим новую таблицу `admin_user`, то она будет сохранять всю информацию, связанную с определением таблицы, в файл `admin_user.frm`.

Мы можем посмотреть информацию, связанную с таблицей, с помощью команды `SHOW TABLE STATUS`. Попробуем выполнить эту команду для таблицы `admin_user` и вытащить информацию.

```
mysql> SHOW TABLE STATUS LIKE 'admin_user' \G;
*************************** 1. row ***************************
 Name: admin_user
 Engine: InnoDB
 Version: 10
 Row_format: Dynamic
 Rows: 2
 Avg_row_length: 8192
 Data_length: 16384
 Max_data_length: 0
 Index_length: 16384
 Data_free: 0
 Auto_increment: 3
 Create_time: 2017-06-19 14:46:49
 Update_time: 2017-06-19 15:15:08
 Check_time: NULL
 Collation: utf8_general_ci
 Checksum: NULL
 Create_options:
 Comment: Admin User Table
1 row in set (0.00 sec)
```

Эта команда в поле `Engine` показывает, что таблица хранится в подсистеме хранения данных InnoDB. Есть другая информация, которую можно использовать для других целей, в частности количество строк, длина индекса и т. д.

Подсистема хранения данных помогает обрабатывать различные операции SQL для различных типов таблиц. Каждая подсистема хранения имеет свои преимущества и недостатки. Выбор подсистемы хранения всегда будет зависеть от потребностей. Важно понимать особенности каждой подсистемы хранения и выбирать наиболее подходящую для ваших таблиц, чтобы максимизировать производительность базы данных. В MySQL всякий раз, когда мы создаем новую таблицу, подсистемой хранения данных по умолчанию является InnoDB.

Мы можем сказать, что сервер MySQL использует архитектуру, работающую в режиме самонастройки (plug and play), потому что мы можем легко загружать и выгружать подсистемы хранения данных с сервера MySQL. Все поддерживаемые подсистемы хранения можно увидеть с помощью команды `SHOW ENGINES`. Она предоставит достаточно информации о том, поддерживается ли подсистема хранения данных сервером MySQL или нет, какая подсистема хранения данных используется сервером MySQL по умолчанию. Давайте выполним эту команду и вытащим эту информацию.

```
mysql> SHOW ENGINES \G;
*************************** 1. row ***************************
 Engine: InnoDB
 Support: DEFAULT
 Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
 XA: YES
 Savepoints: YES
*************************** 2. row ***************************
 Engine: MRG_MYISAM
 Support: YES
 Comment: Collection of identical MyISAM tables
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 3. row ***************************
 Engine: MEMORY
 Support: YES
 Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 4. row ***************************
 Engine: BLACKHOLE
 Support: YES
 Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 5. row ***************************
 Engine: MyISAM
 Support: YES
```

```
 Comment: MyISAM storage engine
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 6. row ***************************
 Engine: CSV
 Support: YES
 Comment: CSV storage engine)
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 7. row ***************************
 Engine: ARCHIVE
 Support: YES
 Comment: Archive storage engine
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 8. row ***************************
 Engine: PERFORMANCE_SCHEMA
 Support: YES
 Comment: Performance Schema
Transactions: NO
 XA: NO
 Savepoints: NO
*************************** 9. row ***************************
 Engine: FEDERATED
 Support: NO
 Comment: Federated MySQL storage engine
Transactions: NULL
 XA: NULL
 Savepoints: NULL
 9 rows in set (0.00 sec)
```

InnoDB

В MySQL 8 подсистема хранения данных InnoDB используется по умолчанию и является наиболее широко применяемой из всех других доступных подсистем хранения. Подсистема InnoDB была выпущена вместе с MySQL 5.1 как плагин в 2008 году, и она рассматривается как подсистема хранения по умолчанию, начиная с версии 5.5 и выше. Поддержка подсистемы хранения InnoDB была перенята корпорацией Oracle в октябре 2005 года у финской компании Innobase Oy.

Таблицы InnoDB поддерживают ACID-совместимые фиксации транзакций, откат и возможности аварийного восстановления для защиты пользовательских данных. InnoDB также поддерживает блокировку на уровне строк, что помогает улучшить параллелизм и производительность. InnoDB хранит данные в кластеризованных индексах, чтобы уменьшить операции ввода-вывода для всех запросов SQL на выборку данных на основе первичного ключа. InnoDB также поддерживает ограничения внешнего ключа, которые обеспечивают лучшую целостность данных в базе данных. Максимальный размер таблицы InnoDB может масштабироваться до 256 Тб, что должно быть вполне достаточным во многих случаях использования больших данных.

Важные замечания по InnoDB

Выполнение простого запроса, такого как `SELECT count(*) FROM [имя таблицы]`, без наличия индексов будет очень медленным, поскольку, для того чтобы получить данные, он выполняет полное сканирование таблицы. Если вы хотите часто применять запрос на количество данных к таблице InnoDB, предлагается создавать триггеры на операции вставки и удаления, после чего можно увеличивать или уменьшать счетчики, когда записи вставляются или удаляются, что может помочь вам достигнуть лучшей производительности.

Дамп содержимого MySQL, который используется для создания резервной копии, работает с InnoDB слишком медленно. Во время выполнения команды `mysqldump` можно включить флаги `--opt--compress`, которые фактически сжимают данные, прежде чем делать дамп содержимого вашей базы данных/таблицы MySQL.

InnoDB – это подсистема хранения данных с **мультиверсионным управлением параллелизмом** (MVCC), которая хранит информацию о старых версиях измененных строк, чтобы поддержать функционал транзакций и отката, что оказывается весьма кстати в целях поддержания целостности данных или в случаях аварийного прекращения работы.

Для оптимизации производительности таблицы InnoDB ниже приведено несколько параметров, которые мы можем использовать в настройках `my.cnf`. Однако они зависят от вашей среды и баз данных.

- `innodb_open_files = 300`: определяет максимальное количество файлов, которые она может держать открытыми при работе с режимом `innodb_file_per_table`.
- `innodb_buffer_pool_size = 128M`: задает размер пула в памяти, который может использоваться для кеширования индексов и табличных данных. Это один из важных аспектов настройки таблиц InnoDB. Это значение можно увеличить в зависимости от размера оперативной памяти на сервере.
- `innodb_thread_concurrency = 8`: этот параметр используется для нескольких параллельных потоков, которые будут использоваться для обработки запроса и зависят от числа доступных ЦП.

MyISAM

Подсистема хранения данных MyISAM использовалась по умолчанию для MySQL вплоть до версии 5.5 1. В отличие от InnoDB, таблицы подсистемы хранения данных MyISAM не поддерживают ACID-совместитмость. Таблицы MyISAM поддерживают только блокировку уровня таблицы, поэтому таблицы MyISAM небезопасны для транзакций. Таблицы MyISAM оптимизированы для сжатия и скорости. MyISAM обычно используется, когда вам нужно иметь в основном операции чтения с минимальными транзакционными данными. Максимальный размер таблицы MyISAM может достигать 256 Тб, что помогает в таких случаях, как анализ данных.

Важные примечания относительно таблиц MyISAM

Подсистема хранения данных MyISAM поддерживает полнотекстовое индексирование, которое может помочь в сложных операциях поиска. С помощью полнотекстовых индексов можно индексировать данные, хранящиеся в типах данных `BLOB` и `TEXT`. Мы подробно рассмотрим полнотекстовое индексирование в главе 3 «*Индексирование данных для высокопроизводительных запросов*».

Из-за низких накладных расходов MyISAM использует более простую структуру, которая обеспечивает хорошую производительность; однако это не сильно помогает для получения хорошей производительности, когда есть потребность в лучшем параллелизме и случаях использования, которые не нуждаются в тяжелых операциях чтения. Наиболее распространенной проблемой производительности MyISAM является блокировка таблицы, которая может задерживать ваши параллельные запросы в очереди. Это происходит, когда она блокирует таблицу для любой другой операции до тех пор, пока более ранняя операция не будет выполнена.

Таблица MyISAM не поддерживает транзакции и внешние ключи. Судя по всему, из-за этих ограничений вместо таблиц MyISAM теперь системные таблицы схемы MySQL 8 используют таблицы InnoDB.

Memory

Подсистема хранения в памяти (подсистема оперативного хранения данных) обычно называется подсистемой хранения данных на основе кучи. Она используется для чрезвычайно быстрого доступа к данным. Эта подсистема хранения содержит данные в оперативной памяти, поэтому ей не нужны операции ввода-вывода. Поскольку она хранит данные в оперативной памяти, все данные теряются при перезапуске сервера. Такая подсистема в основном используется для временных таблиц или таблицы подстановки. Эта подсистема поддерживает блокировку на уровне таблицы, которая ограничивает параллелизм с высокой частотой записи.

Ниже приведены важные примечания об оперативных таблицах Memory.

- Оперативная таблица хранит данные в оперативной памяти, которая имеет очень ограниченный объем; если вы попытаетесь записать слишком много данных в оперативную таблицу, она начнет свопить данные на диск, и тогда вы потеряете преимущества подсистемы хранения данных в памяти.
- Оперативные таблицы не поддерживают типы данных `TEXT` и `BLOB`; такие типы данных могут не потребоваться, так как таблицы имеют ограниченную емкость.
- Эта подсистема хранения может использоваться для кеширования результатов; таблицы поиска, например, или почтовые индексы и названия штатов.
- Оперативные таблицы поддерживают индексы на основе B-дерева и хеш-индексы.

Archive

Эта подсистема хранения данных используется для хранения больших объемов исторических данных без каких-либо индексов. Архивные таблицы не имеют ограничений по объему хранимых данных. Архивная подсистема хранения данных оптимизирована для операций с высокой частотой вставки, а также поддерживает блокировку на уровне строк. Такие таблицы хранят данные в сжатом и малом форматах. Архивная подсистема не поддерживает операции `DELETE` или `UPDATE`; она разрешает только операции `INSERT`, `REPLACE` и `SELECT`.

Blackhole

Эта подсистема хранения данных принимает данные, но их не сохраняет. Вместо сохранения данных она отбрасывает (уничтожает) их после каждой вставки.

В следующем ниже примере показана работа таблицы BLACKHOLE:

```
mysql> CREATE TABLE user(id INT, name CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.07 sec)
mysql> INSERT INTO USER VALUES(1,'Kandarp'),(2,'Chintan');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
mysql> SELECT * FROM USER;
Empty set (0.00 sec)
```

И какой тогда прок от такой подсистемы хранения данных? Зачем кому-то ее использовать? Зачем запускать запрос INSERT, который ничего в таблицу не вставляет?

Эта подсистема хранения полезна для репликации с большим количеством серверов. Подсистема хранения данных Blackhole работает в качестве фильтрующего сервера между ведущим и ведомым серверами, который не хранит никаких данных, но который применяет только правила replicate-do-* и replicate-ignore-* и пишет двоичные журналы. Эти двоичные журналы используются для выполнения репликации на ведомых серверах. Мы обсудим это подробно в главе 6 «*Репликация для построения высокодоступных решений*».

CSV

Подсистема хранения данных CSV хранит данные в файлах в формате .csv, используя формат с разделением значений запятыми. Эта подсистема извлекает данные из базы данных и копирует их в .csv. Если вы создаете CSV-файл из электронной таблицы и копируете его на сервер папок данных MYSQL, она может читать данные с помощью запроса SELECT на выборку данных. Аналогичным образом, если вы записываете данные в таблицу, внешняя программа может их прочитать из CSV-файла. Эта подсистема хранения данных используется для обмена данными между программным обеспечением или приложениями. Таблица CSV не поддерживает индексирование и разделение. Чтобы избежать ошибок при создании таблицы, все столбцы в подсистеме хранения данных CSV должны быть определены с атрибутом NOT NULL.

Merge

Эта подсистема хранения данных также называется подсистемой хранения MRG_Myisam. Эта подсистема хранения объединяет все данные в одну таблицу MyISAM и использует ее для ссылки на единственное представление. В таблицах объединения/слияния все столбцы перечисляются в том же порядке. Эти таблицы хороши для сред объединения баз данных.

В следующем ниже примере показано, как создавать таблицы MERGE:

```
mysql> CREATE TABLE user1 (id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,name
CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE user2 (id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,name
CHAR(20)) ENGINE=MyISAM;
```

```
mysql> INSERT INTO user1 (name) VALUES ('abc'),('xyz');
mysql> INSERT INTO user2 (name) VALUES ('def'),('pqr');
mysql> CREATE TABLE user (id INT NOT NULL AUTO_INCREMENT,name CHAR(20),
INDEX(id))ENGINE=MERGE UNION=(user1,user2);
```

Как правило, эта подсистема используется для управления таблицами, связанными с журналом регистрации событий. В отдельных таблицах MyISAM можно задавать различные месяцы журналов и объединять эти таблицы с помощью подсистемы хранения данных MERGE.

Таблицы MyISAM имеют ограничение по объему хранения для операционной системы, но коллекция таблиц MyISAM (MERGE) не имеет таких ограничений. Таким образом, использование подсистемы MERGE позволит вам разделять данные на многочисленные таблицы MyISAM, что может помочь в преодолении ограничений по объему хранения.

С помощью подсистемы MERGE трудно выполнять разделение, следовательно, таблицами MERGE оно не поддерживается, и мы не можем реализовать раздел на таблице MERGE или любой таблице MyISAM.

Federated

Подсистема интегрированного хранения данных FEDERATED позволяет создавать одну базу данных на нескольких физических серверах. Она открывает клиентское соединение с другим сервером и выполняет запросы к таблице, получая и отправляя строки по мере необходимости. Первоначально она рекламировалась как конкурентная функциональность, которая поддерживала многие корпоративные проприетарные серверы баз данных, такие как Microsoft SQL Server и Oracle, но это всегда было натяжкой, мягко говоря. Хотя казалось, что в ней задействовалось много гибкости и хитрых приемов, она оказалась источником многих проблем и по умолчанию деактивирована. Однако мы можем ее активировать, запустив двоичный файл сервера MySQL с параметром `--federated`.

Давайте создадим таблицу `FEDERATED`.

```
CREATE TABLE user_federated (
 id INT(20) NOT NULL AUTO_INCREMENT,
 name VARCHAR(32) NOT NULL DEFAULT '',
 PRIMARY KEY (id),
 INDEX name (name))
ENGINE=FEDERATED DEFAULT CHARSET=latin1
CONNECTION='mysql://remote_user:[password]@remote_host:port/federated/table
';
```

В поле CONNECTION содержится следующая ниже информация для вашей справки:

- `remote_user`: имя пользователя удаленного сервера MySQL;
- `password`: пароль удаленного сервера MySQL;
- `remote_host`: имя хоста удаленного сервера;
- `port`: номер порта удаленного сервера;
- `federated`: имя базы данных удаленного сервера;
- `table`: имя таблицы базы данных удаленного сервера.

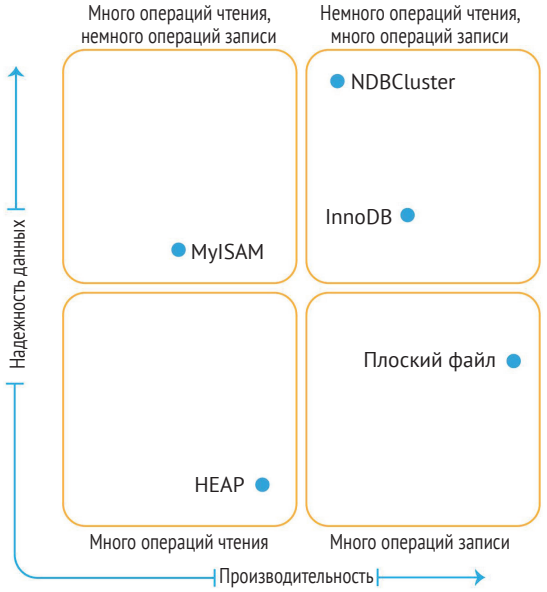
NDB Cluster

NDB Cluster (или просто NDB) – это подсистема хранения прямо в оперативной памяти, обеспечивающая высокую доступность и сохраняемость данных.

Кластерная подсистема хранения данных NDB Cluster может конфигурироваться с помощью ряда параметров аварийного переключения и балансировки нагрузки, но проще всего начать с подсистемы хранения на уровне кластера. NDB Cluster использует подсистему хранения NDB и содержит полный набор данных, который зависит только от других наборов данных, доступных в кластере.

Кластерная часть NDB Cluster настроена независимо от серверов MySQL. В NDB Cluster каждая часть кластера считается узлом.

Следующая ниже схема поможет вам понять, какую подсистему хранения данных вам нужно использовать для ваших потребностей:



Каждая подсистема хранения данных имеет свое преимущество и удобство использования:

- **поисковый механизм**: NDBCluster;
- **транзакционные данные**: InnoDB;
- **сеансовые данные**: MyISAM or NDBCluster;
- **локализованные вычисления**: Memory;
- **словарь**: MyISAM.

Теперь у вас есть более четкое представление о различных подсистемах хранения данных вместе с различными случаями использования, которые помогут вам выбрать свою подсистему в зависимости от ваших потребностей. Давайте рассмотрим операторы обработки данных, используемые для извлечения, сохранения и обновления данных.

Оператор SELECT в MySQL 8

Оператор SELECT используется для извлечения данных из одной или многочисленных таблиц:

```
SELECT поле1, поле2, поле3 from имя_таблицы [WHERE предложение] [GROUP BY {имя_столбца}]
[HAVING условие_where] [ORDER BY {имя_столбца} {ASC | DESC}, ...] [LIMIT{OFFSET M}{LIMIT N}]
```

Это универсальная синтаксическая конструкция, используемая для получения данных из одной таблицы:

- поля поле1 и поле2 являются именами столбцов таблицы. Для извлечения всех столбцов из таблицы можно использовать выражение *;
- имя_таблицы задает имя таблицы, из которой необходимо извлечь данные;
- оператор WHERE может использоваться для задания любого условия в одном и нескольких столбцах;
- функция GROUP BY используется с агрегатными функциями для группирования результирующих наборов;
- предложение HAVING необходимо после GROUP BY для фильтрации на основе условий для группы строк или агрегатов. Если мы используем предложение HAVING без GROUP BY, оно будет действовать аналогично оператору WHERE;
- предложение ORDER BY используется для сортировки результирующих наборов таблицы по возрастанию или убыванию;
- LIMIT используется для ограничения количества строк, возвращаемых оператором SELECT.

Давайте рассмотрим каждый элемент оператора.

Оператор WHERE

Следующий ниже фрагмент кода содержит универсальную синтаксическую конструкцию для запроса SELECT с оператором WHERE:

```
SELECT поле1, поле2, поле3, ..., поле_N from имя_таблицы1, имя_таблицы2
[WHERE условие1 [AND [OR]] условие2...
```

Оператор WHERE является необязательной частью запроса SELECT. Для указания условий можно использовать операторы AND или OR. Оператор WHERE также может использоваться с запросом DELETE и UPDATE, который мы обсудим в ближайшее время в этой главе.

Ниже приведен список операторов, используемых с оператором WHERE для задания условий. Чтобы понять эти операции, давайте рассмотрим пример схемы таблицы. Мы создадим таблицу users с упомянутой здесь схемой, имеющей поля id, first_name, last_name, address, city, state, zip, login_attempts, contact_number, email, username и password:

```
CREATE TABLE `users` (
 `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `first_name` varchar(255),
 `last_name` varchar(255),
 `address` varchar(255),
 `city` varchar(50),
```

```
 `state` varchar(2),
 `zip` varchar(10),
 `login_attempts` int(10),
 `contact_number` varchar(20),
 `email` varchar(191),
 `username` varchar(191),
 `password` varchar(255),
 PRIMARY KEY (`id`)
) ENGINE=InnoDB PRIMARY KEY (`id`) AUTO_INCREMENT=0 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_
unicode_ci;
```

Равно и не равно

Оператор равенства (=) проверяет, равны ли значения двух полей. Если они совпадают, то условие становится истинным, и оператор извлекает значение для дальнейшей обработки. Если они не совпадают, то условие должно содержать оператор неравенства (<>). Оно будет извлекать данные на основе условия, которое не совпадает.

Например, следующий ниже запрос используется для получения всех записей с городом, соответствующим значению New York:

```
SELECT * FROM `users` WHERE `city` = 'New York';
```

Больше и меньше

Оператор больше, чем (>) проверяет, больше ли значение левого поля, чем значение правого поля. Если да, то условие становится истинным. Оператор меньше, чем (<) проверяет, меньше ли значение левого поля, чем значение правого поля. Мы также можем использовать операторы >/< и оператор равенства вместе.

Например, следующий ниже запрос используется для получения всех записей с количеством попыток входа в систему больше, чем 2:

```
SELECT * FROM `users` WHERE `login_attempts` > 2;
```

LIKE

Оператор LIKE предоставляет простой способ поиска записей в столбце с различными шаблонами. В запросе можно использовать подстановочные символы для построения различных шаблонов. В основном используется два вида подстановочных символов. Давайте рассмотрим каждый из них на примере.

- % (процент): используйте этот подстановочный символ для поиска ноль или более любых символов. Предположим, что мы хотим отыскать пользователей, чье имя начинается с «a». Тогда мы можем применить этот подстановочный символ, как показано в приведенном ниже запросе.

  ```
  select * from users where username like 'a%';
  ```

  В случае если мы хотим найти пользователей, чье имя начинается с «a» и заканчивается на «s», то запрос с подстановочным символом % будет таким:

  ```
  select * from user where username like 'a%s';
  ```

- _ (подчеркивание): используйте этот подстановочный символ там, где нужно отыскать записи с шаблоном, где в том месте, где мы указали подчерки-

вание (_), может иметься любой символ. Предположим, мы хотим отыскать пользователей, чье имя заканчивается на dmin, и мы не уверены в первом символе. Следовательно, приведенный ниже запрос будет искать результаты, где первый символ имени пользователя может быть любым, но он должен заканчиваться на dmin.

```
select * from users where username like '_dmin';
```

Важно помнить, что он будет учитывать ровно один символ для одного подчеркивания. Следовательно, в этом случае пользователь с именем пользователя как «aadmin» не будет рассматриваться, потому что в запросе указан всего один подстановочный символ с подчеркиванием.

IN/NOT IN

Оператор IN используется для сравнения нескольких значений в операторе WHERE. Например, следующий ниже запрос используется для поиска всех пользователей, имеющих город new york или chicago:

```
select * from users where city IN ('new york','chicago')
```

Оператор NOT IN работает наоборот, например чтобы найти всех пользователей, у которых нет ни города new york, ни города chicago, используется:

```
select * from users where city NOT IN ('new york','chicago');
```

BETWEEN

Оператор BETWEEN может использоваться в том случае, когда мы хотим получить записи, которые входят в определенный диапазон. Этот диапазон может быть любым, таким как текст, даты или цифры. Предположим, мы хотим отыскать пользователей, дата создания записи о которых находится между 1 июля и 16 июля 2017 года. Тогда приведенный ниже запрос с предложением BETWEEN может нам помочь.

```
select * from users where created_at BETWEEN '2017-07-01 00:00:00' AND '2017-07-16 00:00:00';
```

Подобным образом мы можем также задавать диапазон в числах для поиска пользователей, которые принадлежат к этому конкретному диапазону. Например, если мы хотим получить студентов, чьи оценки находятся в диапазоне от 70 до 80, то оператор BETWEEN может быть применен для определения диапазона.

Предложение ORDER BY

Предложение ORDER BY помогает в получении записей в упорядоченном виде. Оно обеспечивает сортировку данных по определенному столбцу в порядке возрастания или убывания. По умолчанию сортировка выполняется в порядке возрастания, но мы также можем явно указать способ сортировки с помощью ключевых слов ASC и DESC. Если мы зададим ASC, то оно будет сортировать в порядке возрастания, в то время как ключевое слово DESC будет сортировать в порядке убывания. Ниже приведен запрос, который найдет всех пользователей и выведет их в порядке возрастания по столбцу city.

```
SELECT * FROM users ORDER BY city ASC;
```

Кроме того, с помощью предложения ORDER BY можно сортировать в нескольких столбцах. Как показано в приведенном ниже запросе, где мы упорядочиваем по столбцам city и username, мы можем передать несколько столбцов.

```
SELECT * FROM users ORDER BY city, username;
```

Предложение LIMIT

Используя предложение LIMIT, мы можем получить только некоторое количество строк из больших блоков данных. Это помогает ограничить количество строк, возвращаемых в результирующем наборе. Предположим, что в таблице существуют тысячи строк, но нам требуется только 10 записей, тогда вместо извлечения тысяч записей это предложение помогает получить лишь 10 записей. Это действительно помогает настраивать производительность при поиске больших наборов данных.

С предложением LIMIT можно передавать один или два аргумента. В случае двух аргументов один из них будет смещением, которое задает сдвиг первой возвращаемой строки от начала. В то время как второй аргумент будет количеством, которое задает максимальное количество строк, которые будут возвращены. Этот аргумент должен быть нулевым или положительным. Взгляните на приведенный ниже запрос, где мы извлекли 10 записей из таблицы пользователей, начиная с 5-й строки.

```
SELECT * FROM users limit 5, 10;
```

Если с предложением LIMIT указать всего один аргумент, то аргумент будет считаться количеством строк. Например, следующий ниже запрос используется для извлечения 10 строк из таблицы пользователей:

```
SELECT * FROM users limit 10;
```

На данный момент мы видели получение данных из одной таблицы; если мы хотим получить данные из нескольких таблиц, используются ключевые слова JOIN и UNION.

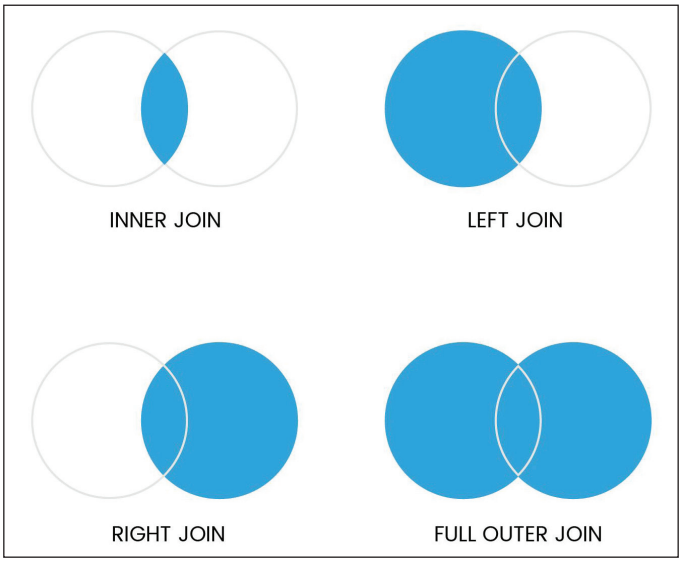
Операции соединения SQL

Операция соединения используется для извлечения данных из нескольких таблиц. Например, если есть две таблицы order и customer и мы хотим получить данные, то это можно сделать с помощью предложения JOIN.

Различные типы предложения JOIN следующие:

- INNER JOIN: внутреннее соединение возвращает только те записи, значения которых совпадают в обеих таблицах;
- CROSS JOIN: перекрестное соединение возвращает только те записи, которые имеют совпадающие значения в левой или правой таблице;
- LEFT JOIN: левое соединение возвращает все записи из левой таблицы и только совпадающие записи из правой таблицы;
- RIGHT JOIN: правое соединение возвращает все записи из правой таблицы и только совпадающие записи из левой таблицы.

На следующем ниже рисунке показан краткий пример для рассмотрения:



INNER JOIN

Внутреннее соединение возвращает записи, совпадающие в обеих таблицах. Например, ниже приведена таблица заказов `order`, используемая в приложениях электронной коммерции:

order_id	customer_id	order_amount	order_date	ship_id
1001	2	7	2017-07-18	3
1002	37	3	2017-07-19	1
1003	77	8	2017-07-20	2

Далее приведем пример таблицы клиентов `customer`, используемой в приложениях электронной коммерции, которая содержит данные о клиентах:

customer_id	name	country	city	postal_code
1	Alfreds Futterkiste	Germany	Berlin	12209
2	Ana Trujillo	Mexico	Мйxico D.F.	05021
3	Antonio Moreno	Mexico	Мйxico D.F.	05023

Следующий ниже запрос будет получать все записи заказа со сведениями о клиенте. Поскольку идентификаторы клиентов 37 и 77 отсутствуют в таблице `customer`, будут получены только совпадающие строки, за исключением идентификаторов клиентов 37 и 77:

```
SELECT order.order_id, customer.name
FROM order
INNER JOIN customer ON order.customer_id = customer.customer_id;
```

LEFT JOIN

Левое соединение `LEFT JOIN` извлекает все записи из левой таблицы и только совпадающие записи из правой таблицы. Если применить пример таблиц клиентов и заказов с левым соединением, то оно будет извлекать все записи из таблицы за-

казов order, даже если в правой таблице нет совпадений (customer). Чтобы получить все сведения о заказах для клиента, можно использовать следующий ниже запрос:

```
SELECT order.order_id, customer.name
FROM order
LEFT JOIN customer ON order.customer_id = customer.customer_id;
```

RIGHT JOIN

Правое соединение RIGHT JOIN извлекает все записи из правой таблицы и общие записи из левой таблицы. Если применить пример таблиц клиентов и заказов с правым соединением, то оно будет извлекать все записи из таблицы клиентов customer, даже если в левой таблице (order) нет совпадений. Чтобы получить все сведения о клиентах с заказом, можно использовать следующий ниже запрос:

```
SELECT order.order_id, customer.name
FROM order
RIGHT JOIN customer ON order.customer_id = customer.customer_id;
```

CROSS JOIN

Перекрестное соединение CROSS JOIN возвращает все записи, в которых есть совпадение в левой или правой записи таблицы. Если мы возьмем пример с таблицами заказов и клиентов, то оно вернет пять строк со сведениями о клиентах и заказах:

```
SELECT order.order_id, customer.name FROM order OUTER_JOIN customer ON
order.customer_id = customer.customer_id;
```

UNION

Ключевое слово UNION используется для объединения результатов нескольких запросов SELECT в одну таблицу. Каждый оператор SELECT, используемый в запросе UNION, должен иметь одинаковое количество столбцов, а для каждого столбца требуется одинаковый тип данных. Имена столбцов из первого запроса SELECT используются в качестве столбцов для возвращаемых результатов.

Например, приведем пример таблицы сотрудников Employee:

employee_id	name	city	postal_code	country
1	Robert	Berlin	12209	Germany
2	Mac	Мйxico D.F.	5021	Mexico
3	Patel	Мйxico D.F.	5023	Mexico

Далее приведем пример таблицы подрядчиков Contractor:

contractor_id	name	city	postal_code	country
1	Dave	Berlin	12209	Germany
2	Robert	Мйxico D.F.	5021	Mexico
3	Patel	Мйxico D.F.	5023	Mexico

Как показано в следующем ниже примере, имена столбцов из первого запроса SELECT используются в качестве столбцов для возвращаемых результатов:

```
SELECT 'Employee' As Type, name, city, country FROM Employee
UNION
SELECT 'Contractor', name, city, country FROM Contractor;
```

Следующий ниже пример объединения UNION удаляет повторяющиеся строки из результата; если мы не хотим удалять повторяющиеся строки, можно применить ключевое слово ALL:

```
SELECT country FROM Employee UNION SELECT country FROM Contractor;
```

Этот запрос возвращает две строки из предыдущей таблицы – Germany и Mexico. Чтобы получить все строки из обеих таблиц, можно использовать ключевое слово ALL следующим образом:

```
SELECT country FROM Employee UNION ALL SELECT country FROM Contractor;
```

Подзапрос

Оператор SELECT внутри оператора SELECT называется подзапросом. Этот подзапрос может применяться для выбора столбца или в условных предложениях. Подзапрос также может быть вложен в другой запрос типа INSERT, UPDATE или DELETE.

Следующий ниже запрос возвращает множество пользователей, которые находятся в Ahmedabad, используя вложенный запрос:

```
SELECT firstName, lastName FROM users
WHERE city IN
(SELECT city FROM cities WHERE city = 'Ahmedabad');
```

Оптимизация запросов SELECT

Запросы SELECT используются для получения данных на динамических вебстраницах, поэтому настройка этих операторов обеспечивает хорошую производительность, что очень важно. Ниже приведены некоторые рекомендации по оптимизации запросов.

- Убедитесь, что в таблицах есть индексы. Индексы всегда помогают ускорить фильтрацию и получение результатов. Индексы можно задавать в операторе WHERE запроса SELECT.
- Индексы также минимизируют количество полных сканирований в больших таблицах.
- Настройка *буферного пула InnoDB*, *кеша ключей MylSAM* и *кеша запросов MySQL* помогает кешировать результаты, которые приведут к более быстрым извлечениям повторяющихся результатов. Размер кеш-памяти можно настроить так, чтобы она обеспечивала более быстрый доступ, предоставляя результаты только из кеша.
- Отрегулируйте размер и свойства используемых MySQL областей памяти, чтобы кешировать буферный пул InnoDB, кеш ключей MylSAM и кеш запросов MySQL. Это помогает выполнять повторные запросы SELECT быстрее.
- Мы должны использовать оператор WHERE вместо HAVING, если мы не используем предложение GROUP BY или другие агрегатные функции, такие как COUNT(), MIN(), MAX(), AVG() и т. д.
- Используйте инструкцию EXPLAIN для анализа запроса с операторами WHERE, JOIN и индексами.

Теперь давайте рассмотрим, в чем заключается полезность инструкции EXPLAIN для оптимизации производительности запросов:

```
EXPLAIN SELECT * FROM `sales_order_item` i
INNER JOIN sales_order o ON i.order_id = o.entity_id
INNER JOIN catalog_product_entity p ON p.sku = i.sku
INNER JOIN customer_entity c ON c.entity_id = o.customer_id
WHERE i.order_id = 42
```

Приведенный ниже результат является выводом выполненного вопроса:

```
*************************** 1. row ***************************
 id: 1
 select_type: SIMPLE
 table: o
 partitions: NULL
 type: const
possible_keys: PRIMARY,SALES_ORDER_CUSTOMER_ID
 key: PRIMARY
 key_len: 4
 ref: const
 rows: 1
 filtered: 100.00
 Extra: NULL
*************************** 2. row ***************************
 id: 1
 select_type: SIMPLE
 table: c
 partitions: NULL
 type: const
possible_keys: PRIMARY
 key: PRIMARY
 key_len: 4
 ref: const
 rows: 1
 filtered: 100.00
 Extra: NULL
*************************** 4. row ***************************
 id: 1
 select_type: SIMPLE
 table: p
 partitions: NULL
 type: ref
possible_keys: CATALOG_PRODUCT_ENTITY_SKU
 key: CATALOG_PRODUCT_ENTITY_SKU
 key_len: 195
 ref: ecommerce.i.sku
 rows: 1
 filtered: 100.00
 Extra: Using index condition
 4 rows in set (0.01 sec)
```

В приведенных выше результатах `possible_keys` показывает, какие индексы применяются к этому запросу, а `KEY` говорит о том, какой из них был фактически использован. В нашем примере каждое соединение `JOIN` использует индексный ключ; путем создания индекса мы должны попытаться избежать значения `NULL` ключа. Поле `ROWS` сообщает нам, сколько строк сканируется в запросе, выполняе-

мом для идентификации, и это число должно быть уменьшено для лучшей производительности запроса, поскольку это минимизирует передачу данных.

Операторы INSERT, REPLACE и UPDATE в MySQL 8

Операторы INSERT, REPLACE и UPDATE используются для вставки или изменения данных в базе данных. Давайте рассмотрим их по очереди.

INSERT

Оператор INSERT вставляет новые строки в существующую таблицу; ниже приведена синтаксическая конструкция добавления новых строк в таблицу:

```
INSERT [INTO] имя_таблицы [(поле1, поле2...)] {VALUES | VALUE}
({значение поля1}, значение поля2, ...),(...),...
```

Следующий ниже пример вставляет новых пользователей в таблицу пользователей:

```
INSERT INTO user (first_name,last_name,email,password) VALUES
("Robert","Dotson","RuthFDotson@teleworm.us", "17500917b5cd7cefdb57e87ce73485d3"),
("Ruth","Uselton","DarronAUselton@armyspy.com", "17500917b5cd7cefdb57e87ce73485d3");
```

UPDATE

Оператор UPDATE модифицирует данные в существующей таблице; ниже приведена синтаксическая конструкция добавления новых строк в таблицу:

```
UPDATE имя_таблицы SET имя_поля1="значение" [, имя_поля2="значение"] ...
[WHERE условие_where]
[ORDER BY ...]
[LIMIT количество_строк]
```

Следующий ниже запрос UPDATE используется, чтобы обновить имя в таблице пользователей:

```
UPDATE users SET first_name = "abc" WHERE email =
"DarronAUselton@armyspy.com";
```

REPLACE

Оператор REPLACE() заменяет все вхождения подстроки в строке. Оператор REPLACE можно использовать с операторами UPDATE или SELECT.

```
REPLACE(строка, поисковая_строка, заменить_на)
```

Следующий ниже запрос MySQL заменяет все вхождения K на SA в пределах столбца country из таблицы users для тех строк, в которых во время выполнения запроса SELECT значение столбца country является INDIA:

```
SELECT city,country, REPLACE(country,'K','SA') FROM user WHERE
country='INDIA';
```

Следующий ниже запрос MySQL заменяет все вхождения INDIA на UK в пределах столбца country:

```
UPDATE user SET country = REPLACE(country, 'INDIA', 'UK');
```

Транзакции в MySQL 8

Транзакция – это логическая единица, состоящая из одного или нескольких запросов INSERT, UPDATE и DELETE. Транзакция полезна, когда требуется выполнить несколько операций с базой данных. Либо все успешные изменения будут завершены и зафиксированы, либо неудачные будут отменены, либо ошибки, возникшие во время исполнения, заставят выполнить откат транзакций. Подсистема хранения данных InnoDB поддерживает откат транзакций. Цель модели хранения InnoDB состоит в том, чтобы объединить лучшие свойства мультиверсионной базы данных с двухфазной блокировкой. InnoDB по умолчанию выполняет блокировку на уровне строки и выполняет запросы как неблокирующие для обеспечения последовательных чтений, тем самым гарантируя лучшую производительность и целостность данных.

Давайте взглянем на пример, чтобы понять, где полезно применять транзакцию. Рассмотрим банковскую базу данных. Предположим, что клиент банка хочет перевести деньги со своего счета на другой счет. Как правило, с помощью транзакции запросы SQL будут разделены на пять частей:

- начало транзакции;
- дебетование со счета клиента;
- кредитование на другом счете клиента;
- запись в журналы транзакций;
- завершение и фиксация транзакции.

Запрос SQL с банковским счетом клиента будет выглядеть следующим образом:

```
UPDATE sb_accounts SET balance = balance - 1000 WHERE account_no = 932656;
```

Запрос SQL по кредитованию другого счета клиента будет такой:

```
UPDATE sb_accounts SET balance = balance + 1000 WHERE account_no = 932600;
```

Запрос SQL для записи сведений о транзакции в журнал будет следующей:

```
INSERT INTO journal VALUES
(100896, 'Transaction on Benjamin Hampshair a/c', '26-AUG-08' 932656,
932600, 1000);
```

Запрос SQL для завершения транзакции будет такой:

```
COMMIT WORK;
```

Если происходит какая-нибудь ошибка в промежутках между всеми этими шагами, к примеру сервер недоступен, то произойдет откат транзакции, что означает, что запросы на вставку и обновление будут отменены.

Агрегирование данных в MySQL 8

Обычно бывает так, что нужные данные не всегда имеются в базе данных. Чтобы генерировать данные, такие как получение общего количества заказов или суммарной цены всех заказов, MySQL предоставляет много встроенных агрегатных функций, которые можно использовать, чтобы выполнять вычисления на загруженных данных при помощи запросов MySQL. Агрегатными значениями могут

быть суммы, количества, средние значения и т. д. Агрегатные функции связаны с получением данных из одного столбца таблицы, выполнением вычислений по нескольким строкам и возвратом требуемого значения в качестве результата.

Согласно стандарту ISO, в основном имеются следующие ниже агрегатные функции:

- минимум;
- максимум;
- среднее число;
- количество;
- сумма.

Важность агрегатных функций

На предприятиях на каждом уровне существует несколько требований к каждой из диверсифицированных функций для получения агрегированных данных с целью их визуализации. Топ-менеджеры компаний будут заинтересованы в том, чтобы иметь целостное представление обо всей организации; однако менеджер будет ограничен знанием конкретных величин отдела. Человек не обязательно должен нуждаться в таких подробностях. Все это может иметь вид различных вариантов продольного и поперечного анализа на каждом уровне, а также многочисленных расчетов.

Мы можем сделать это с помощью агрегатных функций. Рассмотрим пример базы данных портала электронной коммерции. Портал электронной коммерции может нуждаться в различных данных, таких как средняя цена, общее количество товаров, общая сумма продаж товара, максимальная и минимальная закупочные цены; они в значительной степени не ограничены. Однако их можно проанализировать продольно и поперечно для получения дальнейших подробностей, основываясь на потребностях предприятия.

Прежде чем мы перейдем к подробному рассмотрению каждой функции, важно разобраться в некоторых предложениях SQL.

Предложение GROUP BY

Предложение `GROUP BY` используется для группировки записей с одинаковыми значениями. Оно будет формировать группу общих значений по одному или нескольким столбцам и вернет единый результат. Оно широко применяется там, где мы хотим использовать некоторые агрегатные функции, такие как `COUNT`, `MAX`, `MIN`, `SUM` и `AVG`. Это предложение следует за оператором `WHERE`, как показано в приведенной ниже синтаксической конструкции запроса `SELECT`.

```
SELECT поле(поля) FROM имя_таблицы WHERE условие GROUP BY
имя(имена)_столбца(ов) ORDER BY имя(имена)_столбца(ов);
```

Рассмотрим пример, где требуется получить общее количество компаний в каждой стране.

```
SELECT COUNT(companyID), country FROM Companies GROUP BY country;
```

Предложение HAVING

Предложение `HAVING` используется для получения результатов с условием, основанным на различных агрегатных функциях. Предположим, если мы хотим по-

лучить записи, которые зависят от некоторых операций, таких как SUM, AVERAGE, COUNT, MAX, MIN и т. д., то это предложение нам поможет. Это предложение удаляет ограничение оператора WHERE, поскольку оператор WHERE не может использоваться с агрегатной функцией. Посмотрите на синтаксическую конструкцию, приведенную ниже, в которой предложение HAVING расположено в запросе SELECT.

```
SELECT имя(имена)_столбца(ов) FROM имя_таблицы WHERE условие GROUP BY
имя(имена)_столбца(ов) HAVING условие ORDER BY имя(имена)_столбца(ов);
```

Рассмотрим пример, в котором извлекается общее количество компаний в каждой стране. Это относится только к стране, где существует более 3 компаний.

```
SELECT COUNT(companyId), country FROM Companies GROUP BY Country
HAVING COUNT(companyId) > 5;
```

Минимум

Агрегатная функция MIN возвращает наименьшее значение из загружаемых данных.

Ниже приведен пример поиска самой низкой цены товара из таблицы products путем фильтрации столбца price:

```
SELECT MIN(price) lowest FROM products;
```

Максимум

Агрегатная функция MAX возвращает наибольшее значение из загружаемых данных.

Ниже приведен пример поиска самой высокой цены товара из таблицы products путем фильтрации столбца price:

```
SELECT MAX(price) highest FROM products;
```

Среднее значение

Агрегатная функция AVG возвращает среднее значение из загружаемых данных, которое игнорирует значение NULL.

Ниже приведен пример поиска средней цены товара из таблицы products путем фильтрации столбца price:

```
SELECT AVG(price) average FROM products;
```

Количество

Агрегатная функция COUNT возвращает количество загружаемых данных. Ниже приведен пример поиска количества товаров из таблицы products:

```
SELECT COUNT(*) AS Total FROM products;
```

Сумма

Агрегатная функция SUM возвращает сумму из загружаемых данных, которая игнорирует значение NULL.

Ниже приведен пример поиска общей суммы продаж каждого товара из таблицы заказов orders, сгруппированных по коду товара productcode:

```
SELECT productcode,sum(price * quantity)
total FROM orders GROUP by productcode;
```

С помощью соединений вы также можете увидеть более подробную информацию, такую как название товара.

Ниже приведен пример поиска общей суммы продаж каждого товара из таблицы заказов `orders`, сгруппированных по коду товара `productcode`. Для того чтобы получить название товара, мы проверяем совпадение по названию товара в таблице товаров `products`:

```
SELECT P.productCode, P.productName, SUM(price * quantity) total FROM
orders O INNER JOIN products P ON O.productCode = P.productCode GROUP by
productCode ORDER BY total;
```

JSON

В MySQL 5.7 была введена функциональность JSON, которая позволяет получать результаты набора данных в формате данных JSON, виртуальных столбцах, и ориентировочно 15 функций SQL, которые позволяют отыскивать и использовать данные JSON на стороне сервера. В MySQL 8 существуют дополнительные агрегатные функции, которые могут использоваться в объектах/массивах JSON для представления загруженных данных более оптимизированным способом. Ниже приведены две агрегатные функции JSON, которые были введены в MySQL 8:

- `JSON_OBJECTAGG()`
- `JSON_ARRAYAGG()`

Давайте возьмем пример платформы электронной коммерции, где мы создадим немного таблиц и данные, чтобы разобраться в обеих этих функциях.

Сначала создадим таблицу для товаров, которая будет иметь следующий вид:

```
CREATE TABLE `products` ( `id` int(11) NOT NULL AUTO_INCREMENT,
`pname` varchar(75) DEFAULT NULL,
`pmanufacturer` varchar(75) DEFAULT NULL,
`pprice` int(10) DEFAULT NULL,
PRIMARY KEY (`id`)) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Теперь давайте составим таблицу для атрибутов:

```
CREATE TABLE `attr` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`pname` varchar(120) DEFAULT NULL,
`pdesc` varchar(256) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Теперь мы почти закончили создание таблиц; последней является таблица для стоимости, как показано в следующем ниже фрагменте:

```
CREATE TABLE `value` ( `p_id` int(11) NOT NULL, `attr_id` int(11) NOT NULL, `values` text,
PRIMARY KEY (`p_id`,`attr_id`)) ENGINE=InnoDB
DEFAULT CHARSET=latin1;
```

Теперь нам нужно несколько записей в таблицах, на которые мы будем ссылаться в примерах с JSON.

Давайте вставим данные в таблицы, которые мы создали, – `attribute`, `value` и `products`, – как показано в следующем ниже фрагменте:

```
INSERT INTO attr(id, pname) VALUES (1, "color"),
(2, "material"),
(3, "style"),
(4, "bulb_type"),
(5, "usage"),
(6, "cpu_type"),
(7, "cpu_speed"),
(8, "weight"),
(9, "battery_life"),
(10, "fuel_type");

INSERT INTO products(id, pname, pmanufacturer, pprice) VALUES
(1, "LED Desk Lamp", "X", 26);

INSERT INTO products(id, pname, pmanufacturer, pprice) VALUES
(2, "Laptop", "Y", 800);

INSERT INTO products(id, pname, pmanufacturer, pprice) VALUES
(3, "Grill", "Z", 300);

INSERT INTO value VALUES
(2, 1, "blue"),
(2, 6, "quad core"),
(2, 7, "3400 mhz"),
(2, 8, "2,1 kg"),
(2, 9, "9h");

INSERT INTO value VALUES (3, 1, "black"),
(3, 8, "5 kg"),
(3, 10, "gas");
```

JSON_OBJECTAGG

Агрегатная функция `JSON_OBJECTAGG()` считывает два столбца или выражения и показывает их соответственно как ключ и значение в одном объекте JSON. Ключ `NULL` вызовет ошибку, в то время как повторяющиеся ключи будут игнорироваться. Ниже приведен соответствующий пример:

```
mysql> SELECT id, col FROM table;
+------+-----------------------------------+
| id   | col                               |
+------+-----------------------------------+
| 1    | {"key1": "value1", "key2": "value2"}  |
| 2    | {"keyA": "valueA", "keyB": "valueB"}  |
+------+-----------------------------------+
2 rows in set (0.00 sec)
mysql> SELECT JSON_OBJECTAGG(id, col) FROM table;
+-----------------------------------------------------+
| JSON_OBJECTAGG(id, col)                             |
+-----------------------------------------------------+
| {"1": {"key1": "value1", "key2": "value2"}, "2": {"keyA": "valueA",
"keyB": "valueB"}}                                    |
+-----------------------------------------------------+
row in set (0.00 sec)
```

Теперь давайте подключимся к нашим демонстрационным данным; вам нужно извлечь все товары, которые состоят из всех атрибутов ключей и значений в качестве объекта JSON, задействовав данные из таблиц `products`, `attr` и `value` с помощью агрегатной функции `JSON_OBJECTAGG`. Вы, должно быть, заметили, что таблица `products` имеет структурированные данные, и если мы поближе рассмотрим таблицы `attr` и `value`, то увидим, что они состоят из полуструктурированных данных. Ниже приведен пример вывода из агрегатной функции `JSON_OBJECTS`:

```
SELECT JSON_OBJECT("key", p.id,
                   "title", p.pname,
                   "manufacturer", p.pmanufacturer,
                   "price", p.pprice,
                   "specifications", JSON_OBJECTAGG(a.name, v.value))
                   as product
FROM products as p JOIN value as v ON p.id=v.p_id JOIN attr as a ON
a.id=v.attr_id GROUP BY v.p_id;
+-------------------------------------------------+
|product                                          |
+-------------------------------------------------+
  "key": 1,
  "price": 26,
  ";title": "LED Desk Lamp",
  "manufacturer": "X",
  "specifications": {
    "color": "black",
    "style": "classic",
    "usage": "Indoor use only",
    "material": "plastic",
    "bulb_type": "LED"
  "key": 2,
  "price": 800,
  "title": "Laptop",
  "manufacturer": "Y",
  "specifications": {
    "color": "blue",
    "weight": "2,1 kg",
    "cpu_type": "quad core",
    "cpu_speed": "3400 mhz",
    "battery_life": "9h"
  "key": 3,
  "price": 300,
  "title": "Grill",
  "manufacturer": "Z",
  "specifications": {
    "color": "black",
    "weight": "5 kg",
    "fuel_type": "gas"
+-------------------------------------------------+
3 rows in set (0,01 sec
```

JSON_ARRAYAGG

Агрегатная функция `JSON_ARRYAGG` считывает столбец или выражение в качестве аргумента и отображает выходные данные в виде одного массива JSON после

агрегирования результатов. Ниже приведен пример упорядочивания элементов в массиве, который будет не определен.

```
mysql> SELECT col FROM table;
+-----------------------------------+
| col                               |
+-----------------------------------+
| {"key1": "value1", "key2": "value2"} |
| {"keyA": "valueA", "keyB": "valueB"} |
+-----------------------------------+
2 rows in set (0.00 sec)
myysql> SELECT JSON_ARRAYAGG(col) FROM table;
+-----------------------------------------------------+
| JSON_ARRAYAGG(col)                                  |
+-----------------------------------------------------+
| [{"key1": "value1", "key2": "value2"}, {"keyA": "valueA", "keyB":
"valueB"}]                                           |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Теперь давайте подключимся к нашим демонстрационным данным; вам нужно извлечь все товары, которые состоят из возможных атрибутов ключей и значений, задействуя данные из таблиц `products`, `attr` и `value` с помощью агрегатной функции `JSON_ARRYAGG`. Ниже приведен пример вывода из агрегатной функции `JSON_ARRAYAGG`:

```
SELECT p.id, JSON_ARRAYAGG(a.pname) as prod_attributes
FROM products as p JOIN value as v ON p.id=v.p_id
JOIN attr as a ON a.id=p_id
GROUP BY v.p_id;
+----+-------------------------------------------------------------+
| id | prod_attributes                                             |
+----+-------------------------------------------------------------+
| 1  | ["color", "style", "usage", "material", "bulb_type"]        |
| 2  | ["cpu_type", "weight", "color","cpu_speed", "battery_life"] |
| 3  | ["color", "fuel_type", "weight"]                            |
+----+———————————————————————————————————————+
3 rows in set (0,01 sec)
```

Давайте представим, что нужно создать новую схему с использованием этих функций. Это поможет, если вы хотите выполнить миграцию на новую схему. Ниже приведен пример запроса и его результатов:

```
CREATE TABLE all_product AS (SELECT p.id, p.pname, p.pmanufacturer,
p.pprice,JSON_OBJECTAGG(a.name, v.value) as semi_data
FROM products as p JOIN value as v ON p.id=v.p_id
JOIN attr as a ON a.id=v.attr_id GROUP BY v.p_id);
SELECT * FROM all_product;
+---+--------------+---+----+---------------------------------
----------------------------------+
+---+--------------+---+----+---------------------------------
----------------------------------+
| 1 | LED Desk Lamp | X | 26 | {"color": "black", "style":
"classic", "usage": "Indoor use only", "material": "plastic",
"bulb_type":"LED"} |
```

```
| 2 | Laptop        | Y | 800 | {"color": "blue", "weight":
"2,1 kg", "cpu_type": "quad core", "cpu_speed": "3400 mhz", "battery_life": "9h"} |
| 3 | Grill         | Z | 300 | {"color": "black", "weight":
"5 kg", "fuel_type": "gas"}
+---+--------------+---+----+---------------------------------
----------------------------------+
3 rows in set (0,00 sec)
```

Резюме

Теперь вы познакомились с разными подсистемами хранения данных в СУБД MySQL 8 и их применениями. Мы выяснили, как создавать таблицы, изменять существующие таблицы и как добавлять/удалять столбцы в существующую таблицу. Мы также разобрались в том, как удалять таблицы и базы данных. Мы подробно познакомились с различными критериями фильтрации, имеющимися в MySQL. После этого мы изучили различные формы соединений, доступные в MySQL 8, и то, как использовать соединения в запросе для извлечения содержательных данных из базы данных в одном запросе. Мы исследовали то, как в MySQL работают транзакции базы данных, и важность транзакций в реальных сценариях.

Наконец, мы выяснили, как в MySQL используются агрегатные функции, и познакомились с использованием агрегатных предложений `GROUP BY` и `HAVING` и различных агрегатных функций, таких как `MIN()`, `MAX()`, `AVG()`, `SUM()` и `COUNT()`, а также с тем, как хранить информацию JSON в базе данных.

В следующей главе вы узнаете, каким образом в MySQL 8 работает индексирование, новый функционал, связанный с индексированием, различные типы индексирования и как использовать индексирование в ваших таблицах.

Глава 3

Индексирование данных для высокопроизводительных запросов

В предыдущей главе вы научились применять запросы к данным, хранящимся в базе данных MySQL. Вы изучили различные синтаксические конструкции запроса SELECT, способы соединения таблиц и применения агрегатных функций к таблице.

В этой главе вы узнаете, что такое индексирование и различные типы индексов:

- индексирование в MySQL;
- типы индексов MySQL;
- индексирование данных JSON.

Давайте предположим, что у нас есть таблица базы данных, которая имеет более чем 50 миллионов записей с адресами электронной почты, и вы хотите получить одну запись из этой таблицы. Теперь, если вы напишете запрос, чтобы получить адрес электронной почты, MySQL должен будет выполнить проверку в каждой строке в поисках значений, соответствующих вашему запрашиваемому адресу электронной почты. Если MySQL требуется одна микросекунда для сканирования одной записи, то потребуется около пяти секунд, чтобы загрузить всего одну запись, и, поскольку количество записей в таблице увеличивается, затрачиваемое время также будет увеличиваться экспоненциально, что повлияет на производительность!

К счастью, нынешний функционал технологий реляционных баз данных имеет решения для быстрого извлечения информации из больших наборов данных. На таблицах могут быть определены индексы, которые быстро извлекают информацию из таблицы без сканирования всей таблицы. Индексы фактически являются указателями на строки в таблице. Определение индекса на таблице не требует каких-либо изменений в фактическом определении таблицы; индексы могут создаваться независимо. Мы можем определить один или несколько индексов или один индекс на нескольких столбцах, исходя из того, как мы будем манипулировать и извлекать данные из таблиц.

На приведенном ниже рисунке показан простой пример индексирования записи. В правой таблице записи хранятся в соответствии с тем, как вставляется новая запись; каждая новая запись вставляется в конец таблицы. Поэтому если вы хоти-

те найти записи, содержащие определенное слово, подсистема хранения данных должна просканировать всю таблицу, чтобы получить соответствующие записи. Однако если эти данные правильно отсортированы внутри столбцов, подсистемы хранения данных смогут легко находить записи и возвращать результаты.

Индексирование в MySQL

MySQL поддерживает различные индексы на своих таблицах для быстрого доступа к записи. Прежде чем определять индексы на таблицах, важно знать, когда нужно индексировать данные, и не менее важно выбрать правильное поле для создания индексов.

В следующем списке показано, когда следует использовать индексирование:

- когда требуется группирование на основе определенного столбца;
- когда требуется сортировка по определенному столбцу;
- когда необходимо найти минимальное и максимальное значения в таблице;
- когда имеется большой набор данных и нужно часто отыскивать несколько записей на основе определенных условий;
- когда необходимо выполнять запрос, имеющий соединение между двумя или несколькими таблицами.

Индексные структуры

В разных методах индексирования используются разные структуры для хранения индексной информации в базе данных:

- индексы на основе битовых карт;
- разреженные индексы;
- плотные индексы;
- индексы на основе B-деревьев;
- хеш-индексы.

Давайте быстро пройдемся по каждому из этих индексов.

Индексы на основе битовых карт

Как следует из названия, индекс на основе битовой карты хранит информацию о столбцах в виде бита. Битовая индексация используется, когда в столбцах присутствует низкая мощность в уникальных значениях. Например, поле `gender` в таб-

лице может иметь два значения – мужской или женский. И все строки таблицы могут иметь значения мужской или женский.

Поэтому вопрос в том, что на самом деле хранится в таких индексах? Каждая строка в таблице индекса на основе битовой карты имеет два значения – ноль или один, и каждая строка таблицы индекса имеет столько битов, сколько имеется строк в таблице. Для каждого уникального значения столбцов существует один индекс на основе битовой карты. В общем случае индекс на основе битовой карты использует битовый массив, а во время поиска для возврата записей использует побитовую операцию.

Мы не будем подробно обсуждать индексы на основе битовой карты; в настоящее время таких индексов MySQL 8 не поддерживает.

Разреженные индексы

В разреженном индексе все искомые записи в базе данных не хранятся; в индексе хранится только одна запись в расчете на блок в отсортированном порядке. Если запрашиваемая запись в индексе не найдена, база данных выполнит последовательный поиск в этом блоке записей, чтобы найти запрашиваемые записи. Это поведение медленнее, чем плотный индекс, поскольку разреженный индекс хранит только несколько записей в индексе, а размер индекса ниже, чем плотный индекс.

Плотные индексы

В плотном индексе имеется индекс для каждой искомой записи в базе данных. Поскольку все искомые записи доступны непосредственно из индекса, он становится очень быстрым, но будет потреблять гораздо больше дискового пространства из-за его одноуровневого поведения.

Индексы на основе B-деревьев

B-дерево является наиболее широко используемой структурой индексирования в СУБД. Подсистема хранения баз данных InnoDB в MySQL использует индексные структуры на основе B-деревьев. B-дерево – это сбалансированная древовидная структура данных, в которой данные хранятся в отсортированном порядке в узле самого низкого уровня. Вследствие своей структуры алгоритм B-дерева, выполняющий поиск в структуре данных B-дерева, возвращает результаты за логарифмическое время. Все значения в индексах B-дерева хранятся на одном уровне.

Узел самого высокого уровня в структуре данных B-дерева называется корневым узлом, тогда как самый низкий узел называется листовым узлом. Корневой узел может указывать или ссылаться на листовой узел. Ссылка на листовой узел содержит указатели на листовой узел или на другой узел, который ссылается на листовой узел. Листовые узлы хранят фактические данные, тогда как узлы, ссылающиеся на листовые узлы, содержат указатели на другой нелистовой узел либо фактические данные.

В структуре данных B-дерева каждый корневой, листовой или нелистовой узел состоит из двусвязного списка. Ссылочный или внутренний узел состоит из минимального ключа дочерней страницы, на которую ссылаются, и номера дочерней страницы. Поскольку каждый узел в B-дереве является двусвязным списком, каждый узел также содержит ссылку на следующий и предыдущий узлы на том же уровне.

Уровень внутреннего узла или дочернего узла увеличивается, как только исчерпывается все пространство в листовом узле. Когда все листовые узлы заполнены, создается новая страница, и корневой узел указывает на новую страницу, и дочерний узел разделяется на два.

При создании индекса в базе данных значения ключа сортируются и индексируются в структуре B-дерева. Когда ключ создается для более чем одного столбца, во время поиска MySQL начинает искать в индексной таблице с самого левого столбца индекса.

Давайте применим индексы на столбцах `first_name` и `last_name` в таблице пользователей `users`, которую мы создали в главе 2 «*Методы запроса данных в MySQL 8*»:

```
CREATE INDEX idx_firstname_lastname ON users(first_name, last_name);
```

Итак, предположим, нам нужны все записи пользователей с именем `Vivek` и фамилией `Patel`:

```
SELECT * from users where first_name = 'Vivek' and last_name = 'Patel';
```

Когда мы применим приведенный выше запрос `SELECT`, индексирование MySQL на основе B-дерева сначала будет искать в столбце `first_name` совпадение записей со значением `Vivek` и затем во втором столбце, `last_name`, совпадение записей со значением `Patel`:

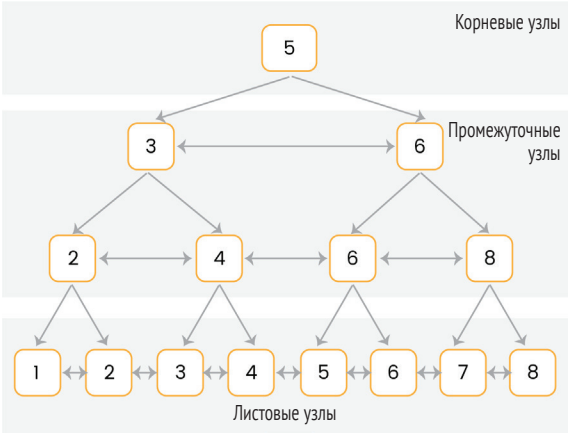
```
SELECT * from users where last_name = 'Patel' and first_name = 'Vivek';
```

С другой стороны, если оператор `SELECT` применить с обратным условием, то MySQL не будет искать данные в индексной таблице, а сделает полное сканирование таблицы, которое может оказаться ресурсоемким в зависимости от ваших данных.

Поэтому очень важно создавать индексы в зависимости от часто используемого запроса. Мы разберемся в особенностях индексирующей таблицы подробнее, когда позже в этой главе увидим различные типы индексирования, используемые в MySQL 8.

Индексы на основе B-дерева используются в MySQL для операций сравнения, таких как =, >, <, >= и =<.

На следующем ниже рисунке показан пример того, как выглядит индекс на основе B-дерева. Здесь каждый узел на одинаковом уровне связан с другим, а также с промежуточными и листовыми узлами:



Хеш-индексы

Хеш-индексы ведут себя иначе, чем структуры данных в B-дереве. Хеш-индекс работает намного быстрее, чем индексы на основе B-дерева. Для индексации данных используются пары ключ-значение. Хеш-индексы допускают только операторы равенства/неравенства, такие как = или <>, где выполняются лишь сравнения значений. Они не поддерживают операторы, где требуется сравнение диапазонов, такие как >, <, >= или <=. При сравнении диапазонов структура B-дерева выполняется за время log(*n*), тогда как в таких же условиях хеш-индекс может занимать время O(*n*), которое может не подходить для многих случаев использования.

В MySQL 8 хеш-индексы поддерживаются только в подсистемах хранения данных Memory и NDB. Они не поддерживаются в подсистемах хранения данных MyISAM или InnoDB.

Хеш-индекс нельзя использовать в предложении `ORDER BY`. В отличие от B-дерева, где крайние левые столбцы могут использоваться для запроса, хеш-структура использует для поиска всю индексную пару ключ-значение.

Создание или удаление индексов

Процедура создания индекса в MySQL очень похожа на то, как это делается в любых других СУБД. Ниже приведена синтаксическая конструкция, используемая для создания индекса со всеми возможными параметрами:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX имя_индекса
[тип_индекса]
ON имя_таблицы (имя_индексного_столбца, ...)
[параметр_индекса]
[параметр_алгоритма | параметр_блокировки] ...
```

UNIQUE | *FULLTEXT* | *SPATIAL*

В заголовке раздела приведены типы индексов, которые можно определять в MySQL 8. Пространственный индекс `SPATIAL` доступен только в подсистемах хранения данных InnoDB и MyISAM. Определение индекса `SPATIAL` в других подсистемах приведет к ошибке.

имя_индексного_столбца

`Имя_индексного_столбца` – это имя столбца, где мы должны задать индекс, как показано в следующем ниже фрагменте.

```
имя_индексного_столбца: имя_столбца [(длина)] [ASC | DESC]
```

В имени индексного столбца можно определить длину индексного столбца, упорядоченность индекса и необходимость создания индекса в порядке возрастания или убывания. При использовании столбца строкового типа, такого как `char`, `varchar` или `text`, можно определить фактическую длину столбца. Ниже приведен пример запроса для создания индекса на столбце `first_name` таблицы `users` с индексом на первых 15 символах запроса:

```
CREATE INDEX idx_first_name ON users(first_name(15));
```

параметр_индекса

Ниже приведены различные параметры индекса; каждый из них будет объяснен далее.

```
параметр_индекса:
  KEY_BLOCK_SIZE [=] значение |
  тип_индекса |
  WITH PARSER имя_анализатора |
  COMMENT 'строковое_значение' |
  {VISIBLE | INVISIBLE}
    тип_индекса: USING
    {BTREE | HASH}
KEY_BLOCK_SIZE
```

С помощью параметра KEY_BLOCK_SIZE можно задать размер индексного блока. Этот параметр не поддерживается на индексном уровне в подсистеме хранения данных InnoDB.

WITH PARSER

Этот вариант предназначен для полнотекстового индекса и может использоваться для задания алгоритма лексического анализа, доступного в MySQL в качестве плагина полнотекстового поиска.

COMMENT

Для созданного индекса можно указать комментарии. Это делается в пояснительных целях. В MySQL 8 максимальный размер комментария составляет 1,024 символа.

VISIBILITY

Этот параметр служит для изменения или упоминания видимости индекса в таблице. Невидимый индекс рассматривается как скрытый индекс и не используется оптимизатором MySQL для индексирования данных. Он доступен для любой подсистемы хранения данных.

тип_индекса

Тип индекса можно определить в используемой подсистеме хранения данных. Ниже приведен список подсистем хранения данных, в которых можно определить тип_индекса. Тип индекса нельзя использовать с полнотекстовым FULLTEXT или пространственным SPATIAL индексами. Когда используется тип индекса, который не поддерживается MySQL, то MySQL будет в таблице автоматически использовать тип индекса по умолчанию.

Имя подсистем хранения	Тип индекса
InnoDB	BTREE
MyISAM	BTREE
Memory/HEAP	HASH, BTREE
NDB	HASH, BTREE

параметр_алгоритма

Параметр ALGORITHM используется для определения способа перестроения таблицы при изменении индекса. Когда параметр алгоритма задан как INPLACE, используется встроенная методология перестроения индекса, когда определение таблицы модифицируется.

```
параметр_алгоритма:
  ALGORITHM [=] {DEFAULT|INPLACE|COPY}
```

Если параметр ALGORITHM задан как COPY, то при изменении определения таблицы он использует метод внутреннего копирования таблицы и перестроения индексов.

Если используется алгоритм по умолчанию, то MySQL автоматически выберет алгоритм на основе доступного варианта.

параметр_блокировки

Для управления уровнем параллельных операций с таблицей при ее изменении целесообразно использовать предложение LOCK.

```
параметр_блокировки:
   LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE} имя_индексного_столбца
```

Если параметр LOCK задан как SHARED, то во время операций изменения таблиц могут параллельно исполняться другие запросы. Если параметр LOCK имеет значение NONE, то во время операций изменения таблицы разрешены другие операции DDL. Если параметр LOCK задан как EXCLUSIVE, то MySQL сначала будет ожидать до тех пор, пока любые существующие операции чтения/записи не будут завершены, и затем примет блокировку, которая заблокирует дальнейшие операции чтения/записи до тех пор, пока не завершится операция индексирующего запроса.

Ниже приведен простой пример создания и удаления индексов из таблицы базы данных, предполагая, что у нас есть таблица employee, которая имеет столбцы employee_id, firstname, email_address и joining_date (дата приема на работу):

```
CREATE TABLE IF NOT EXISTS `employee` (
 `employee_id` int(11) NOT NULL,
 `firstname` varchar(50) NOT NULL,
 `email_address` varchar(100) NOT NULL,
 `joining_date` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Employee Meta data
information' AUTO_INCREMENT=1;
```

Теперь давайте определим первичный ключ для employee_id, который всегда будет уникальным, не нулевым и никогда не будет повторяться:

```
ALTER TABLE `employee` ADD PRIMARY KEY (`employee_id`);
```

Кроме того, электронный адрес email_address сотрудника всегда должен быть уникальным, с тем чтобы мы могли определить уникальный ключ на столбце email_address:

```
ALTER TABLE `employee` ADD UNIQUE KEY `unique_email_address`
(`email_address`);
```

Нередко нам может понадобиться список сотрудников, отфильтрованных и отсортированных по их дате приема на работу joining_date. Поэтому, чтобы сделать запрос быстрее, мы определяем индекс на столбце даты приема на работу:

```
ALTER TABLE `employee` ADD KEY `idx_joining_date` (`joining_date`);
```

Для исключения индекса из таблицы используется следующая ниже синтаксическая конструкция:

```
DROP INDEX INDEX_NAME on имя_таблицы;
```

Когда следует избегать индексирования

К этому моменту вы познакомились с различными структурами индексирования и с тем, как создавать или удалять индексы в MySQL 8. Индексирование может действительно повысить производительность запроса, но бывают случаи, когда создание индекса не улучшает производительность запроса или даже может повлиять на нее отрицательно.

Когда мощность таблицы очень низка, излишнее создание индекса может на самом деле не иметь большого преимущества. Причина в том, что MySQL выполняет полное сканирование таблицы, если с индексированием он должен просканировать 30% записей. В таких случаях следует избегать создания каких-либо индексов в таблице.

Для создания индекса требуется дополнительное дисковое пространство. Кроме того, когда индекс создан, MySQL должен обновлять таблицу индекса всякий раз, когда строка вставляется, обновляется или удаляется. Таким образом, создание слишком большого количества индексов в таблице может оказать существенное влияние на запросы вставки/удаления, и в результате они замедлятся.

Типы индексов СУБД MySQL 8

В предыдущих разделах вы изучали различные индексные структуры и способы создания индексов в таблице. Теперь давайте подробно рассмотрим все имеющиеся в MySQL индексы и их важность.

Когда вы создаете таблицу в MySQL, существует пять типов вариантов индексов:

- первичный (PRIMARY);
- уникальный (UNIQUE);
- столбцовый (COLUMN);
- полнотекстовый (FULLTEXT);
- пространственный (SPATIAL).

Основываясь на конструкции базы данных, можно выбрать любой из этих индексов в своей таблице; частота используемых в запросе данных и столбцов соответственно поможет определить, где индексы должны применяться.

Определение первичного индекса

Первичный ключ используется для уникальной идентификации каждой строки. Столбец, на котором определен первичный ключ, является уникальным по своей природе и содержит ненулевые значения. В следующем разделе вы узнаете, как использовать первичный ключ и разницу между суррогатным ключом и естественным ключом.

Первичные ключи

Каждая таблица InnoDB содержит один специальный индекс, который уникально определяет каждую строку. Этот столбец также называется кластеризованным индексом. Кластеризованный индекс в большинстве случаев является синонимом первичного ключа.

Всякий раз, когда определяется первичный ключ, MySQL использует его как кластеризованный индекс. Если первичный ключ не определен, то MySQL будет

проверять, не был ли уникальный индекс определен как имеющий ненулевые значения. Если будет найден какой-либо соответствующий столбец, то MySQL будет использовать этот столбец как кластеризованный индекс.

Если нет ни первичного ключа, ни уникального ключа, то MySQL внутренне создаст скрытый столбец с уникальными идентификаторами строки и определит на этом столбце кластеризованный индекс, который MySQL будет использовать внутренне для вторичных индексов. Каждый уникальный идентификатор строки состоит из 6 байт, который увеличивается на основе новых вносимых данных. В идеале на столбце всегда следует определять первичный ключ. Если столбцы не являются уникальными, то на таблице всегда можно создать автоинкрементный числовой первичный ключ, который можно использовать в качестве кластеризованного индекса.

Все индексы, кроме кластеризованных, считаются вторичными. В каждом вторичном индексе хранится один столбец кластеризованного индекса, который помогает быстрее отыскать запись и получить к ней доступ. Это также означает, что если первичный ключ длинный, то вторичный индекс будет потреблять относительно больше дискового пространства.

Естественные ключи против суррогатных ключей

Когда первичный ключ таблицы состоит из фактических данных, он называется естественным ключом. Естественные ключи создаются из одного или нескольких столбцов таблицы.

Иногда, когда во время выполнения требуется генерировать данные, первичный ключ создается с использованием автоинкрементных значений, которые называются **суррогатным ключом**.

Давайте рассмотрим следующий ниже пример, чтобы получше понять этот тип ключа:

```
CREATE TABLE users (
 email_address VARCHAR(255) NOT NULL PRIMARY KEY,
 first_name VARCHAR(100) NULL,
 last_name VARCHAR(100) NULL
) ENGINE=InnoDB;
```

В таблице `users` будет указан следующий формат данных:

email_address	first_name	last_name
User1@mail.com	Firstname1	Lastname1
User2@mail.com	Firstname2	Lastname2
User3@mail.com	Firstname3	Lastname3
User4@mail.com	Firstname4	Lastname4

Здесь мы задали адрес электронной почты в качестве первичного ключа, который является естественным ключом, и индексация в этой таблице поможет при поиске пользователя по первичному ключу.

Что делать, если есть несколько пользователей, с которыми не связан адрес электронной почты? В этом случае мы не сможем использовать первичный ключ на столбце `email_address`. Мы можем создать еще один столбец, который является автоинкрементным и который может быть установлен в качестве первичного ключа. Мы можем установить уникальный ключ на столбце `email_address`, который

будет разрешать нулевые значения внутри поля email_address; однако он будет накладывать ограничение, которое заключается в том, что email_address должен быть уникальным для каждого пользователя. Это поможет удовлетворить требования первичного ключа и обеспечить согласованность данных:

```
CREATE TABLE users (
 user_id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
 email_address VARCHAR(255) NULL DEFAULT NULL,
 first_name VARCHAR(100) NULL,
 last_name VARCHAR(100) NULL,
 UNIQUE(email_address)
) ENGINE=InnoDB;
```

Суррогатные ключи в большинстве случаев являются числовыми и не имеют никакой связи с данными, хранящимися в таблицах, тогда как естественные ключи состоят из одного или нескольких столбцов данных. При поиске суррогатные ключи не предоставляют никаких содержательных данных, но действуют как естественный ключ, который хранится с использованием фактических значений, помогая нам предоставлять соответствующие значения, запрашиваемые из таблицы.

Естественные ключи могут быть нечисловыми и могут использовать более одного столбца, что, очевидно, может занять сравнительно больше места на диске после создания индексов. Использование естественных ключей также может влиять на запросы JOIN для извлечения данных из таблицы, что не является подходящим сценарием при использовании суррогатных ключей.

Уникальные ключи

Уникальный ключ используется для задания ограничений на столбце, где все значения в данном столбце должны быть неповторяющимися. Уникальные ключи, как и первичные ключи, должны иметь уникальные значения. Однако уникальные ключи допускают значения NULL внутри столбцов. Мы можем определить уникальный ключ на группе столбцов или на одном столбце.

Как отмечалось ранее, уникальный ключ может использоваться в качестве кластеризованного ключа, если значения внутри уникального ключа не равны NULL и в таблице не заданы первичные ключи.

Ниже приведен пример создания уникального ключа:

```
ALTER TABLE users ADD UNIQUE INDEX unique_email_address(`email_address`);
```

Удаление уникального ключа аналогично удалению любого столбцового индекса:

```
ALTER TABLE users DROP INDEX unique_email_address;
```

Определение столбцового индекса

Индекс используется для повышения производительности запроса SELECT. Индекс может быть задан для одного или нескольких столбцов. Существует несколько вариантов создания столбцового индекса, исходя из потребностей и использования запроса SELECT. Ниже приведены различные столбцовые индексы, доступные в MySQL 8, и примеры их использования.

Составные индексы в MySQL 8

Очень часто имеется более одного поля, где нам действительно нужно определить индекс для достижения высокой производительности от запроса в крупных наборах данных. Когда индекс создается на нескольких столбцах таблицы, он называется составным индексом. MySQL допускает создание составных индексов максимум на 16 столбцах.

Рассмотрим следующее ниже определение:

```
CREATE TABLE users (
 email_address VARCHAR(255) NOT NULL PRIMARY KEY,
 first_name VARCHAR(100) NOT NULL,
 last_name VARCHAR(100) NOT NULL,
 INDEX idx_user_fullname(first_name,last_name)
);
```

Здесь `idx_user_fullname` – это составной индекс, состоящий из столбцов `first_name` и `last_name`. Теперь, когда мы применим следующий ниже запрос для поиска пользователей по имени, он будет использовать составной индекс `idx_user_fullname`:

```
SELECT * from users where first_name = 'John' and last_name = 'Cena';
```

Если мы применим следующий ниже запрос, то он тоже будет использовать составной индекс:

```
SELECT * from users where first_name = 'John';
```

Рассмотрим в запросе SQL еще один оператор – оператор `OR`:

```
SELECT * from users where first_name = 'John' or first_name = 'Michael';
```

Если мы применим следующий ниже запрос, то он также будет использовать составной индекс.

```
SELECT * from users where first_name = 'John' and (last_name = 'Cena' or
last_name = 'Michael');
```

Все предыдущие запросы для получения желаемых результатов будут возвращать результат, используя составной индекс.

Вместе с тем если при использовании составного индекса следующие ниже запросы применяются с небольшим изменением в порядке следования столбцов в запросе, то для поиска в таблице результатов MySQL не будет использовать индекс и выполнит полное сканирование таблицы. Поэтому, чтобы оптимально использовать преимущества индексирования, рекомендуется обеспечивать синхронизацию структуры базы данных и запросов:

```
SELECT * from users where last_name = 'John' and first_name = 'Cena';
```

Если мы применим следующий ниже запрос, то он тоже будет использовать составной индекс:

```
SELECT * from users where last_name = 'Cena';
```

Рассмотрим в запросе SQL еще один оператор – оператор `OR`:

```
SELECT * from users where last_name = 'Cena' or last_name = 'Stuart';
```

Если мы применим следующий ниже запрос, то он тоже будет использовать составной индекс:

```
SELECT * from users where last_name = 'Cena' and (first_name = 'Alex' or
first_name = John);
```

Покрывающий индекс

Когда индекс создается на столбцах таблицы, которые требуются в запросе, он называется покрывающим индексом. Создание покрывающих индексов может привести к повышению производительности.

Иногда бывает нужен запрос `SELECT`, в котором количество полей в столбцах и количество фильтров одинаковы. В таких случаях, если мы задаем индекс на фильтрующих критериях, результаты могут быть получены непосредственно из индекса. В свою очередь, эта операция может быть очень быстрой и производить быстрый результат.

```
ALTER TABLE users ADD INDEX idx_first_name_last_name (first_name, last_name);
```

Теперь применим следующий ниже запрос `SELECT` к этой таблице `users`:

```
SELECT first_name,last_name from users where first_name = 'John' and last_name = 'Cena';
```

Как мы видим, параметры фильтра и параметры `SELECT` одинаковы. Так что MySQL будет возвращать результат непосредственного из самого индекса.

Невидимые индексы

Как мы узнали в главе 1 «*Введение в большие данные и MySQL 8*», невидимые индексы были введены в MySQL 8. При создании невидимого индекса оптимизатору MySQL будет сообщено, что он не должен использовать этот индекс в запросе:

```
ALTER TABLE users ALTER INDEX email_address INVISIBLE;
```

Приведенный выше запрос сделает индекс `email_address` невидимым. Когда индексы делаются невидимыми, MySQL по-прежнему будет поддерживать индексную таблицу на `email_address` или на операциях вставки/обновления. Невидимые индексы можно использовать в тех случаях, когда мы хотим измерить производительность созданных нами запросов и проверить, используется индексация в запросе или нет.

Первичный ключ таблицы нельзя сделать невидимым. Если первичный ключ в таблице не задан и если на любом другом столбце таблицы задан уникальный ключ, то MySQL рассмотрит уникальный ненулевой столбец как первичный и не допустит невидимое индексирование на этих столбцах.

Нисходящие индексы

Как и невидимые индексы, нисходящие индексы также являются расширенным функционалом MySQL 8 и не присутствовали в более ранних версиях MySQL.

Нисходящие индексы позволяют создавать индекс в обратном порядке, который выполняет запрос очень быстро, когда используется сортировка столбцов по убыванию. До определенной степени он может сократить результат запроса.

Давайте разберемся в порядке убывания более подробно на примере. Рассмотрим следующие ниже данные для таблицы `users`:

email_address	first_name	last_name	registered_date
alex@mail.com	Alex	Lastname1	2017-04-04 11:23:53
john@mail.com	John	Lastname2	2017-03-06 15:14:34
stuart@mail.com	Stuart	Lastname3	2017-03-04 10:12:12
lynda@mail.com	Lynda	Lastname4	2017-04-04 10:23:53

Если мы хотим, чтобы пользователи сортировались в порядке убывания по дате регистрации, но по возрастанию по именам запрос будет следующим:

```
select * from users order by registered_date desc, first_name asc;
```

Теперь положим, что у нас есть индекс, определенный на столбцах `registered_date` и `first_name`:

```
alter table users add key idx_registered_date_name (registered_date desc, first_name asc);
```

В предыдущий запрос мы добавим `explain`:

```
explain select * from users order by registered_date desc, first_name asc;
```

Результат будет следующим:

Id	1
Select type	SIMPLE
table	users
Type	ALL
Possible_keys	NULL
Key	NULL
Key_len	NULL
Ref	NULL
Rows	4
Extra	Using Index

Это всего лишь пример, поэтому мы показали результат приведенной выше инструкции explain. В общем случае MySQL не будет использовать никакого индекса в таблице, если записей очень мало.

Здесь, как мы видим, в строке **Extra** упоминается, что MySQL использует индекс для предложения `ORDER BY`. Это означает, что наш запрос для получения отсортированных записей использует индексирование.

В версиях до MySQL 8 если существует другой порядок сортировки в индексном столбце, то для сортировки данных MySQL будет использовать `file sort`. Для получения списка отсортированных записей метод `file sort` использует полное сканирование таблицы.

Нисходящие индексы поддерживаются только в подсистеме хранения данных InnoDB. Нисходящие индексы нельзя использовать для полнотекстового или пространственного индекса. Вы также не можете определить нисходящие индексы на столбце `FTS_DOC_ID`, который используется для полнотекстового индексирования в подсистеме хранения данных InnoDB.

Определение внешнего ключа в таблице MySQL

Внешний ключ – это ссылка на записи в главной таблице. Хотя внешний ключ не является индексом, важно задать индексы в таблице явным образом, поскольку они могут повысить производительность во время операции соединения таблицы.

Ниже приведена синтаксическая конструкция добавления внешнего ключа при создании новой таблицы:

```
ALTER TABLE имя_таблицы
ADD [CONSTRAINT [символ]] FOREIGN KEY
[index_name] (имя_индексного_столбца, ...)
REFERENCES имя_таблицы (имя_индексного_столбца, ...)
[ON DELETE параметр_ссылки]
[ON UPDATE параметр_ссылки]
параметр_ссылки: RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Следующие разделы являются кратким обзором различных `параметров_ссылки`, допустимых при задании внешних ключей.

RESTRICT

Если в дочерней таблице найдена ссылающаяся строка, то MySQL не разрешит удалять записи из родительской таблицы.

CASCADE

Если из родительской таблицы какие-либо записи удаляются или обновляются, то это вызовет действие `DELETE` или `UPDATE` и для дочерних таблиц. Поэтому если удалить запись из родительской таблицы, то все ссылающиеся записи из всех дочерних таблиц, где параметр ссылки имеет значение `ON DELETE CASCADE`, будут тоже удалены.

SET NULL

При обновлении или удалении любой записи из родительской таблицы значение столбца внешнего ключа будет равно `NULL`.

Когда любая запись из родительской таблицы обновляется или удаляется, то этот параметр установит значение столбца внешнего ключа равным `NULL`.

NO ACTION

Параметр `NO ACTION` похож на параметр `RESTRICT` и не позволяет удалять записи из родительской таблицы, если в дочерней таблице существует какая-либо ссылающаяся строка.

SET DEFAULT

Подсистемы хранения данных InnoDB и NDB не допускают использования параметра ссылки `SET DEFAULT` в табличной синтаксической конструкции.

При определении внешнего ключа типы данных первичного ключа и внешнего ключа должны совпадать. Кроме того, подсистема хранения данных для родительской и дочерней таблиц должна быть одинаковой. Чтобы быстрее выполнить проверку внешнего ключа, необходимо определить индекс на столбце, в котором мы определяем внешние ключи. MySQL не допускает префиксацию индексов на столбцах, где определен внешний ключ. В MySQL 8 мы можем определять первичный ключ и внешний ключ на тех же таблицах, но на разных столбцах.

MySQL поддерживает ссылочную целостность при изменении, удалении или создании новой таблицы. Поэтому, когда вы отбрасываете существующую таблицу, если имеются какие-либо дочерние таблицы, определяющие внешние ключи, то MySQL выдаст ошибку. Кроме того, при воссоздании таблицы, когда имеются какие-либо существующие внешние ключи, определенные на другой таблице,

то имена столбцов и порядок столбцов должны быть такими же, как упомянуто в ограничениях внешнего ключа, иначе MySQL покажет ошибку при создании таблицы.

Перед удалением или созданием таблицы можно установить «foreign_key_checks = 0»; когда foreign_key_checks равняется нулю, MySQL не будет проверять никакие ссылки на исполняемые команды SQL. По умолчанию значение foreign_key_checks равняется единице.

Удаление внешних ключей

Ниже приведен запрос на удаление внешнего ключа из таблицы.

```
ALTER TABLE имя_таблицы DROP FOREIGN KEY символ_внешнего_ключа;
```

Обратите внимание, что удаление внешнего ключа не приведет к удалению из таблицы явно заданных индексов.

Полнотекстовая индексация

Полнотекстовое индексирование используется для поисковых запросов, где необходимо отыскать ключевое слово в предложении или абзаце, хранящихся в таблицах MySQL.

Ниже приведен пример создания таблицы с полнотекстовым индексом:

```
CREATE TABLE blog (
 blog_id INT(11) AUTO_INCREMENT NOT NULL PRIMARY KEY,
 blog_title VARCHAR(255) NOT NULL,
 blog_description TEXT,
 FULLTEXT(blog_title,blog_description)
) ENGINE=InnoDB;
```

А теперь, когда вы хотите найти ключевое слово – предположим, «initial» в таблице blog, используя полнотекстовое индексирование, вы можете использовать следующий ниже запрос:

```
SELECT * FROM blog WHERE MATCH (blog_title, blog_description) AGAINST
("initial" IN NATURAL LANGUAGE MODE);
```

Этот запрос вернет все блоги, в записях которых найдено соответствующее ключевое слово «initial». Когда мы используем соответствующие ключевые слова в условии WHERE, MySQL будет возвращать строки, отсортированные согласно их релевантности искомому ключевому слову. Запись с самой высокой релевантностью будет в результирующем наборе первой, а с самой низкой релевантностью – последней. Для вычисления релевантности MySQL принимает во внимание следующие параметры:

- общее количество слов в строке;
- общее количество уникальных слов в строке;
- общее количество слов в хранящемся полнотекстовом индексе;
- количество строк, совпадающих с искомыми ключевыми словами.

Если вы хотите узнать оценку релевантности строк, возвращаемых MySQL, в именах столбцов можно применить запрос MATCH:

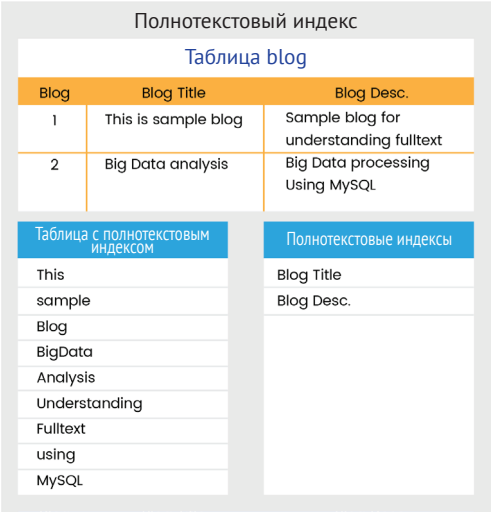
```
SELECT MATCH(blog_title, blog_description) AGAINST ("initial" IN NATURAL
LANGUAGE MODE) as relevant_score FROM blog;
```

В силу сказанного у вас может возникнуть вопрос: что, если мы используем запрос MATCH в столбцах вместе с условием WHERE? Будет ли он выполнять индексный поиск многократно?

Ответ – **нет**. Когда запрос исполняется, оптимизатор MySQL проверяет, используется ли тот же запрос MATCH в условии WHERE. Если да, то индексный поиск будет выполняться всего один раз.

Поскольку полнотекстовый поиск применяется для поиска ключевого слова в тексте, полнотекстовые индексы могут создаваться только для типов данных CHAR, VARCHAR или TEXT. Рекомендуется создавать полнотекстовый индекс после сохранения данных в таблице, а не создавать его во время создания таблицы. Хранение больших данных в таблицах с определенными полнотекстовыми индексами может привести к снижению производительности при выполнении операций вставки.

Ниже приведен простой пример; в нем показано, как полнотекстовый индекс преобразовывает таблично-индексированные столбцы в лексемизированные полнотекстовые индексы:



Естественно-языковой полнотекстовый поиск на InnoDB и MyISAM

В естественно-языковом полнотекстовом поиске столбец, используемый в условии MATCH, должен совпадать со столбцами, которые использовались при создании полнотекстовых индексов. Как мы видели в предыдущем примере, чтобы совпасть с нашим ключевым словом, в запросе с ключевым словом MATCH мы использовали поля blog_title и blog_description. Однако мы не можем использовать blog_title или blog_description по отдельности. Если вы хотите отыскать ключевое слово только в заголовке, то в поле blog_title необходимо создать полнотекстовый индекс.

При использовании естественно-языковых полнотекстовых индексов применяется определенная логика предметной области.

Полнотекстовый поиск имеет предварительно собранный список ключевых слов, которые действуют как стоп-слова. Это означает, что он не будет индексиро-

вать ни одно из этих слов. Например, стоп-слова is, are и т. д. в поисковый индекс не включаются. MySQL поддерживает принятый по умолчанию список ключевых слов, который может быть изменен. MySQL предоставляет параметр `ft_stopword_file`, где вместо использования принятого по умолчанию списка стоп-слов мы можем определить имя файла, который должен использоваться в качестве стоп-слов для более оптимальной настройки в соответствии с вашими потребностями.

Полнотекстовый анализатор MySQL допускает наличие в словах апострофов и подчеркиваний. Это означает, что список слов может в качестве допустимых содержать такие слова, как `mysql_fulltext` или `mysql's featurelist`. В этом случае у нас будет три отдельных слова, которые совпадают, то есть `mysql_fulltext`, `Mysql's` и `featurelist`. Если апострофы используются более одного раза, то они пропускаются и будут рассматриваться как два отдельных слова. Например, `Mysql's` будет считаться двумя отдельными словами – `Mysql` и `s`. Полнотекстовый поиск будет считать словом только `MySQL` для поиска в индексе, так как минимальная длина для полнотекстового поиска по умолчанию составляет четыре символа.

Если имеется двойная кавычка, используемая в начале и конце искомых ключевых слов, то MySQL будет считать их как одиночный литерал и произведет поиск всех ключевых слов в этом литерале.

Если искомые ключевые слова присутствуют больше чем в 50% записей таблицы, то полнотекстовый поиск вернет пустые результаты, поскольку в MySQL установлен порог на естественно-языковой полнотекстовый поиск в размере 50%. Если вы не ожидаете такого поведения и хотите, чтобы отображались все результаты, то вместо этого следует использовать булев поиск, который не имеет порогового значения.

Полнотекстовое индексирование на InnoDB

Когда полнотекстовый индекс определен на любом столбце InnoDB, имеющем тип `char`, `varchar` или `text`, то создается ряд индексных таблиц. Эти таблицы называются вспомогательными таблицами. Всякий раз, когда новая запись вставляется в таблицу MySQL, ее значения в столбцах в зависимости от полнотекстового индекса лексемизируются (разбиваются) на многочисленные слова. Все эти лексемизированные слова вставляются во вспомогательные таблицы со ссылкой на строку таблицы. Там, где входящие данные разделяются и сохраняются, создаются шесть таких вспомогательных таблиц.

Поскольку существует несколько вспомогательных таблиц, в которых хранятся лексемизированные слова, обновление всех этих таблиц при каждой вставке новой записи может оказаться слишком дорогостоящим, так как к этим таблицам несколько раз будут создаваться параллельные запросы. Чтобы избежать этой ситуации, MySQL использует полнотекстовые индексные кеши для кеширования недавно вставленных строк. Вновь вставленные индексные данные первоначально хранятся в кеше памяти. Чтобы ограничить или определить количество данных, хранящихся в памяти, используются параметры MySQL. С целью конфигурирования размера полнотекстового кеша для каждой из таблиц может использоваться параметр `innodb_ft_cache_size`. Чтобы определить общее ограничение кеша, может быть настроен параметр `innodb_ft_total_cache_size`. Когда значение размера кеша достигает или превышает установленный предел, MySQL очистит кеш и вытолкнет кешированные данные в таблицы.

Полнотекстовый поиск в булевом режиме IN BOOLEAN MODE

Использование полнотекстового индексирования в булевом режиме позволяет пользователям отыскивать несколько слов в индексных столбцах с помощью различных операторов. Эти операторы используются в начале или конце слов.

Рассмотрим следующую ниже таблицу `blog`, которая позволит нам лучше разобраться в принципе его работы:

blog_id	blog_title	blog_description
1	Introduction to MySQL 8 for big data	This blog will explain importance of MySQL 8 in bigdata lifecycle
2	Querying techniques in MySQL 8	This blog will explain how to query MySQL 8 data
3	Indexing in MySQL 8	This blog will explain how indexing works in MySQL 8

Показанный ниже запрос будет искать все строки, содержащие слово `Querying` в качестве ключевого слова, но которое не имеет индексации.

```
SELECT * FROM blog where MATCH (blog_title, blog_description) AGAINST
('+Querying -Indexing' IN BOOLEAN MODE);
```

Ниже приведен список операторов, поддерживаемых в полнотекстовых запросах в логическом режиме `IN BOOLEAN MODE`:

+	Этот символ означает, что слова должны присутствовать в строках, возвращаемых запросом. InnoDB поддерживает только начальные символы +, тогда как MyISAM может поддерживать начальные и замыкающие символы +
-	Этот символ означает, что слово не должно присутствовать в строках, возвращаемых запросом. InnoDB поддерживает только начальные знаки, в то время как MyISAM может поддерживать начальные или замыкающие символы
`@distance`	Оператор `@distance` работает только для InnoDB и измеряет расстояние между двумя словами, упомянутыми в поисковом запросе. Можно задать список слов в двойных кавычках (""), за которыми следует оператор `@distance`
~	Знак тильды в качестве начального оператора будет считаться отрицательной оценкой строк, совпадающих со словами. В отличие от оператора –, слова будут рассматриваться в искомых и возвращаемых строках
*	Звездочка считается подстановочным оператором и всегда используется в качестве замыкающего оператора. Когда за ключевым словом следует звездочка, булев поиск будет искать строки, совпадающие со всеми записями, которые начинаются со слова, указанного перед * (звездочкой). Слово не считается стоп-словом, если за ним следует * (звездочка)
()	Круглые скобки используются для определения в поисковом запросе вложенных выражений. В запросе может быть несколько вложенных скобок. Мы вскоре увидим примеры скобок в запросе
><	Символы больше и меньше используются для изменения ранга слова в оценках релевантности. Символ больше увеличивает вес этого слова в поисковом запросе, в то время как символ меньше уменьшает вес этого слова в запросе
""	Двойные кавычки считаются буквальными. При использовании в поисковом запросе двойных кавычек полнотекстовый поиск будет искать в строках все слова, заключенные в двойные кавычки, и точно в том же порядке, как указано. При поиске литералов, как указано в поисковом запросе, символы стоп-слов игнорируются, например `','` или `'.'`

Ниже приведено несколько ключевых свойств полнотекстового индексирования MySQL в булевом режиме:

- полнотекстовый поиск в булевом режиме не содержит порога 50%, как у естественно-языкового полнотекстового индексирования;
- при использовании булева режима полнотекстового индексирования в InnoDB он требует, чтобы все столбцы, используемые в выражении `MATCH`, име-

лись в заданном полнотекстовом индексе. MyISAM может работать в булевом режиме без задания полнотекстового индекса, но эта работа может быть ресурсоемкой;

- при использовании указанных операторов подсистема хранения данных InnoDB принимает одиночные начальные операторы в полнотекстовых запросах. Например, в InnoDB поддерживается `+Querying`, однако `++Querying` в случае InnoDB вызовет синтаксическую ошибку, тогда как MyISAM второй булев оператор автоматически проигнорирует;
- булевы операторы в качестве суффикса не поддерживаются в InnoDB. Например, `Querying-` или `Querying+` в InnoDB не поддерживаются и покажут синтаксическую ошибку;
- в InnoDB и MyISAM используется список стоп-слов, которые можно конфигурировать. Таблицы `innodb_ft_user_stopword_table`, `innodb_ft_enable_stopword` и `innodb_ft_server_stopword_table` используются в InnoDB в качестве списка стоп-слов, в то время как в MyISAM для той же цели используется `ft_stopword_file`.

Различия между полнотекстовой индексацией и запросами like

Запросы `LIKE` обычно используются для нахождения точных совпадений:

```
select * from users where first_name like 'John';
```

Рассмотрим приведенный выше запрос, который используется для поиска всех результатов с точным сопадением `John`. Теперь, если существует индекс, определенный на столбце `first_name`, MySQL будет использовать этот индекс, чтобы найти результаты, совпадающие с `John`.

Рассмотрим следующий ниже запрос; в запросе никакого индекса не будет, поскольку MySQL не использует индексы при запросах, которые имеют подстановочный знак. Поэтому произойдет полное сканирование таблицы, что повлияет на производительность выполнения запроса.

```
select * from users where first_name like '%John%';
```

Рассмотрим следующий далее запрос, который не начинается с подстановочного знака, но содержит подстановочный знак в конце фильтра:

```
select * from users where first_name like 'John Cena%';
```

Этот запрос вернет все результаты, которые начинаются со строки `John Cena`. Вместе с тем никаких результатов не будет, если какой-либо пользователь имеет имя `John` или `Cena`, что может быть как раз тем, что мы ожидаем получить в качестве нашего результата.

Как мы увидели в этом разделе, для полнотекстового поиска мы можем определить полнотекстовые индексы, которые для поиска релевантных записей используют алгоритм на основе лексемизации. Использование полнотекстового поиска с полнотекстовым индексированием широко применяется по причине его производительности и гибкости.

Пространственные индексы

Пространственные индексы отличаются от всех остальных индексов. Они не могут применяться к одномерным данным. Они могут использоваться для индексирования географических объектов, которые, как правило, являются многомерными.

Пространственные индексы для индексирования данных используют структуру данных R-дерево. Начиная с MySQL 5.7 пространственные индексы поддерживаются в подсистемах хранения данных MyISAM и InnoDB.

Ниже приведен список типов пространственных данных, поддерживаемых в MySQL:

- POINT (точка);
- LINESTRING (отрезок);
- GEOMETRY (геометрия);
- POLYGON (многоугольник);
- MULTIPOINT(набор точек);
- MULTILINESTRING (набор отрезков);
- GEOMETRYCOLLECTION (коллекция типов);
- MULTIPOLYGON (набор многоугольников).

Ниже приведен пример того, как создавать пространственный индекс на любом пространственном столбце:

```
CREATE TABLE stationary_map (
 stationary_id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
 stationary_name varchar(255) NOT NULL,
 stationary_location POINT NOT NULL
);
```

Чтобы создать индекс на столбце `stationary_location`, может использоваться следующий ниже запрос:

```
ALTER TABLE stationary_map ADD SPATIAL INDEX(stationary_location);
```

Как показано в следующем ниже фрагменте, пространственный индекс можно удалить точно так же, как и любые другие индексы:

```
DROP INDEX stationary_location ON stationary_map;
```

Давайте теперь рассмотрим индексирование данных JSON, которое играет важную роль, когда мы говорим о больших данных и аналитике больших данных, используя их во многих практических случаях.

Индексирование данных JSON

Большие данные состоят из многочисленных документов, которые хранятся в формате JSON, поэтому необходимо иметь надлежащее индексирование данных JSON. Прямой способ определения индекса на данных JSON отсутствует. Мы всегда можем создавать столбцы из данных JSON, а затем задействовать сгенерированные столбцы для поддержки индексов.

Давайте сначала разберемся, что такое сгенерированные столбцы в MySQL.

Генерируемые столбцы

Генерируемые столбцы обычно не имеют фактической информации, но они хранят информацию, собранную из других столбцов таблицы, используя некоторые выражения или вычисления.

Сгенерированные столбцы можно разделить на два типа: **виртуальные сгенерированные столбцы** и **сохраненные сгенерированные столбцы**.

Виртуальные сгенерированные столбцы

Столбцы, заданные как виртуальные сгенерированные столбцы, не хранят значение фактически, а вычисляются на лету. Если для созданных столбцов не определен тип столбца, то по умолчанию будут использоваться виртуальные столбцы. InnoDB поддерживает вторичные индексы на фактически генерируемых столбцах. Поскольку значения виртуальных столбцов вычисляются на лету, они не требуют дополнительного пространства для их хранения. Однако если индекс создан на виртуальных столбцах (только для InnoDB), индекс потребует места для хранения индексированного значения.

Сохраненные сгенерированные столбцы

Значения сохраненных сгенерированных столбцов вычисляются и сохраняются при вставке новой строки или обновлении существующей строки. Они хранят значения в базе данных, следовательно, требуют дискового пространства. Кроме того, сохраненные сгенерированные столбцы можно индексировать.

Сгенерированные столбцы можно использовать в различных сценариях.

- Выражение может храниться как сгенерированный столбец и индексироваться для повышения производительности запросов. Благодаря этому сгенерированный столбец можно использовать для кеширования выражения. Это выражение также может использоваться как общее сохраненное выражение для нескольких запросов `SELECT`.
- Когда запрос `SELECT` исполняется с оператором `WHERE`, и выражение оператора `WHERE` при этом совпадает со сгенерированными столбцами, сгенерированные столбцы индексируются. Оптимизатор MySQL будет автоматически использовать сгенерированные столбцы, даже если они явно не определены в запросе.
- Сгенерированные столбцы можно использовать для вычисления или оценки значений. Это может быть удобно, в особенности для столбцов с типом JSON.

В MySQL определены конкретные правила использования сгенерированных столбцов:

- допускаются любые детерминированные операторы, литералы или встроенные функции. Функция называется детерминированной, если она может вычислять значения из данных, имеющихся в таблице, и не требует никакой информации от подключенных пользователей;
- для сгенерированного столбца нельзя использовать автоинкрементное свойство;
- использование вложенных запросов, переменных, пользовательских функций, хранимых процедур или переменных в сгенерированных столбцах запрещено.

Ниже приведена синтаксическая конструкция для создания сгенерированных столбцов в MySQL 8:

```
Имя_столбца тип_данных [GENERATED ALWAYS] AS (выражение)
[VIRTUAL | STORED]
[UNIQUE [KEY]] [COMMENT комментарий]
[[NOT] NULL] [[PRIMARY] KEY]
```

Далее приведен пример того, как использовать сгенерированные столбцы.

Рассмотрим таблицу users, в которой в качестве столбцов используются поля first_name и last_name. Итак, если у нас есть высокая частота, которая потребует объединения полей first_name и last_name как полное имя full_name, то вы можете создать сгенерированный столбец в таблице users, который будет хранить это объединенное строковое значение и будет возвращать значение:

```
CREATE TABLE users (
 first_name varchar(100),
 last_name varchar(100),
 full_name varchar(255) AS (CONCAT(first_name,' ',last_name))
);
```

Теперь вы можете использовать прямой запрос SELECT для извлечения полного имени full_name из базы данных:

```
SELECT full_name from users;
```

В InnoDB на виртуальных столбцах можно определять вторичный индекс. Индексы, созданные на виртуальных столбцах, обычно называются виртуальными индексами. В MySQL InnoDB позволяет создавать вторичный индекс на более чем одном виртуальном столбце, или вы можете его определить на смеси виртуальных столбцов и регулярных столбцов.

Во время создания вторичного индекса на виртуальных столбцах потребуется больше операций записи для создания индексов, так как значение виртуального столбца будет вычисляться во время вставки и обновления. Если индекс на виртуальных столбцах не создается, то значение сгенерированных столбцов будет вычисляться во время выполнения, что может увеличить период выполнения запроса.

Определение индексов на JSON

Как мы отмечали ранее, мы можем создавать сгенерированные столбцы из объекта JSON, а затем можем создать индекс для этих сгенерированных столбцов, который ускоряет доступ к данным JSON.

Рассмотрим объект JSON, который выглядит следующим образом. Нам нужно создать объект JSON для сведений о сотрудниках:

```
[ {
'employee_id' : 'emp_0000001',
'employee_first_name' : 'john',
'employee_last_name' : 'carry',
'employee_total_experience_in_years' : '6.5',
'employee_join_date':'2013-03-01',
'employee_email_address':'john_carry@email.com'
},
{
'employee_id' : 'emp_0000002',
'employee_first_name' : 'bill',
'employee_last_name' : 'watson',
'employee_total_experience_in_years' : '9.5',
'employee_join_date':'2010-05-01',
'employee_email_address':'bill_watson@email.com'
},
{
'employee_id' : 'emp_0000003',
```

```
'employee_first_name' : 'sara',
'employee_last_name' : 'perry',
'employee_total_experience_in_years' : '14.5',
'employee_join_date':'2006-07-01',
'employee_email_address':sara_perry@email.com'
}];
```

Давайте составим таблицу, хранящую информацию JSON в базе данных:

```
CREATE TABLE employee (
 employee_detail JSON,
 employee_id varchar(255) GENERATED ALWAYS AS
 (employee_detail->>'$.employee_id'),
 INDEX idx_employee_id(employee_id)
);
```

Теперь давайте вставим объекты JSON, которые мы создали ниже:

```
INSERT INTO employee (employee_detail) values
('{"employee_id" : "emp_0000001","employee_first_name" :
"john","employee_last_name" : "carry","employee_total_experience_in_years"
: "6.5","employee_join_date":"2013-03-01","employee_email_address":"john_carry@email.com"}'),
('{"employee_id" : "emp_0000002","employee_first_name" :
"bill","employee_last_name" : "watson","employee_total_experience_in_years"
:
"9.5","employee_join_date":"2010-05-01","employee_email_address":"bill_watson@email.com"}'),
('{"employee_id" : "emp_0000003","employee_first_name" :
"sara","employee_last_name" : "perry","employee_total_experience_in_years"
:
"14.5","employee_join_date":"2006-07-01","employee_email_address":"sara_perry@email.com"}');
```

А сейчас применим запрос SELECT, используя сгенерированные столбцы:

```
SELECT employee_detail->>'$.employee_join_date' as joining_date FROM
employee WHERE employee_id = 'emp_0000002';
```

В результате выполнения данного запроса будет напечатано 2010-05-01.

Теперь давайте попробуем выполнить предложение DESCRIBE, чтобы проверить, может ли использоваться какой-либо индексный ключ или нет, как показано в примере ниже:

```
DESCRIBE SELECT employee_detail->>'$.employee_join_date' as joining_date
FROM employee WHERE employee_id = 'emp_0000002';
```

id	1
Select_type	SIMPLE
table	employee
partitions	null
type	ref
Possible_keys	idx_employee_id
Key	idx_employee_id
Key_len	103
Ref	Const
Rows	1
Filtered	100
Extra	null

Как мы видим, в этой таблице содержится индекс `idx_employee_id`, который служит для выборки записей. И благодаря ему мы можем выполнять поиск данных JSON намного быстрее, чем обычный поиск записей JSON для крупных наборов данных.

Резюме

Теперь у вас есть хорошее представление о различных типах индексных структур, поддерживаемых в MySQL, и о том, как создавать индекс в MySQL. Вы также познакомились с различными типами индексов, доступными в MySQL 8, такими как кластеризованный индекс, покрывающий индекс, нисходящий индекс и невидимый индекс. У нас есть хорошее понимание полнотекстового индексирования и того, как работают естественно-языковые полнотекстовые индексы и полнотекстовые индексы в булевом режиме.

В следующей главе мы рассмотрим некоторые интересные приемы, позволяющие повысить производительность MySQL, такие как начальная настройка, применение и конфигурирование плагина Memcached (реализующего службу кеширования данных в оперативной памяти на основе хеш-таблицы) с использованием программных интерфейсов Memcached, анализ данных, хранящихся в Memcached, и как использовать программные интерфейсы Memcached в MySQL 8.

Глава 4

Использование Memcached в MySQL 8

В предыдущей главе мы увидели, каким образом в MySQL 8 работает индексирование и как это может сказываться и улучшать производительность. Вы также познакомились с тем, как создавать индексы, и с различными типами индексов, имеющимися в MySQL 8, такими как кластеризованный индекс, покрывающий индекс, нисходящий индекс и невидимый индекс. У нас есть хорошее понимание полнотекстового индексирования и того, как работают естественно-языковые полнотекстовые индексы и полнотекстовые индексы в булевом режиме.

В этой главе мы рассмотрим использование в MySQL 8 плагина Memcached для кеширования данных в оперативной памяти. Ниже перечислены темы, которые будут затронуты:

- настройка Memcached;
- использование Memcached;
- анализ данных, хранящихся в Memcached;
- конфигурация репликации Memcached;
- API Memcached для различных технологий.

Давайте сначала начнем с обзора Memcached в MySQL 8, за которым последует другая подробная информация о Memcached.

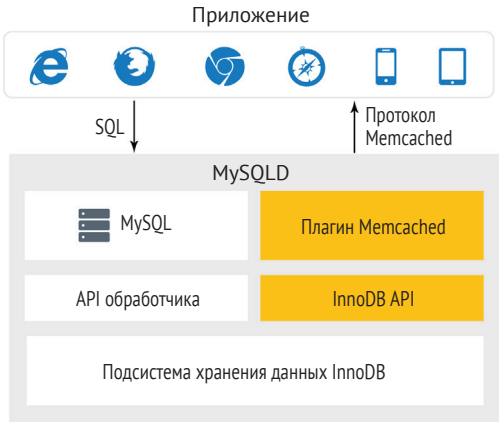
Обзор Memcached

Говоря об обработке больших данных в MySQL, мы, безусловно, должны подумать о **производительности** – о том, как мы справляемся с проблемами производительности при частом размещении и извлечении данных. Один из видных ответов – Memcached, сервер кеширования данных в оперативной памяти на основе хеш-таблицы, который повышает производительность частого размещения и извлечения данных, потому что он пропускает анализатор запросов, оптимизатор SQL и другие части механизма, которые являются ненужными и позволяют сохранять или извлекать данные непосредственно с InnoDB. Использование Memcached позволяет управлять данными гораздо быстрее и делает намного удобнее обработку больших данных.

MySQL 8 предоставляет вам плагин InnoDb Memcached под названием `daemon_memcached`, который способен помочь в управлении данными. Он автоматически сохраняет и извлекает данные из таблицы InnoDB, обеспечивает операции `get`, `set`

и `incr`, которые ликвидируют накладные расходы, влияющие на производительность, за счет пропуска синтаксического анализа SQL, что в итоге ускоряет операции с данными.

Следующая ниже схема поможет вам лучше понять, каким образом при использовании Memcache анализируются запросы:



Как показано в предыдущей схеме, плагин Memcached непосредственно связан с **подсистемой хранения данных InnoDB**, и анализирующие слои SQL не стоят у него на пути. В результате устраняются накладные расходы на извлечение и хранение данных, что приводит к более быстрым результатам, чем традиционный SQL. Он прозрачно записывает данные в таблицу InnoDB, а также легко обрабатывает как неструктурированные, так и структурированные данные. Первоначально данные будут записываться в кеш-память Memcached, а затем автоматически передаваться между памятью и диском. Memcache помогает, когда транзакции базы данных часты, что требует времени для сохранения и извлечения данных; этот плагин обеспечивает эффективный способ обращения с большими данными, достигая лучшей масштабируемости и высокой доступности.

Плагин `daemon_memcached` привязан к серверному процессу MySQL, который избегает сетевых накладных расходов, возникающих из-за синтаксического анализа запросов между различными процессами. Этот плагин может легко выполняться с экземпляром MySQL, поскольку он обеспечивает легкий способ управления памятью, а также относительно меньше потребляет времени центрального процессора.

Плагин Memcached также внутренне управляет буферным пулом, который помогает обслуживать последующие запросы для тех же данных, не входя в подсистему хранения данных InnoDB. Он кеширует частые данные в памяти и предоставляет данные из пула памяти при запросе. Он в основном пропускает выборку данных из таблиц InnoDB, которые в иной ситуации имели бы тяжелые операции ввода-вывода; следовательно, быстрее извлекать данные из памяти буферного пула вместо InnoDB.

InnoDB также имеет возможности обрабатывать композицию и декомпозицию многочисленных значений столбцов в единое значение элемента Memcached. Например, если мы хотим хранить данные в разных столбцах, то можем объединить значения столбцов 1 | 2 | 3 | 4 и хранить в кеше Memcached. InnoDB это поймет

и разобьет значения на основе разделителя, сохранит их в разных столбцах. Этот плагин также имеет функционал сериализации, который преобразовывает двоичные файлы, блоки кода либо любые другие объекты в строковые значения, которые могут быть сохранены, и обеспечивает простоту извлечения таких объектов.

Давайте настроим Memcached и по мере продвижения вперед пройдемся по его конфигурации.

Настройка плагина Memcached

MySQL имеет свой собственный плагин Memcached, `daemon_memcached`, который тесно интегрирован с сервером MySQL. Он улучшает производительность и избегает сетевого трафика, по сравнению с традиционным подходом на основе Memcached.

Традиционно Memcached работал на отдельном порту и в качестве другого процесса; следовательно, это имело бы издержки сетевого взаимодействия между MySQL и Memcached. Вместе с тем этот плагин упростил такое взаимодействие и сделал его более легковесным, так как этот плагин интегрирован непосредственно в сам сервер MySQL.

Ниже перечислены среды, которые поддерживаются этим плагином:

- Linux;
- Solaris;
- OS X.

Давайте посмотрим, как настроить плагин `daemon_memcached` на сервере MySQL. В случае, если мы используем кластерную среду MySQL, нам пришлось бы установить этот плагин на каждом экземпляре сервера MySQL.

Инсталляция

Чтобы активировать функционал Memcached в MySQL, нам сначала нужно инсталлировать библиотеки, от которых зависит плагин Memcached. Для возможности активировать плагин Memcached на сервере должна быть установлена библиотека `libevent`. Следующая ниже команда поможет вам установить `libevent` на сервере Linux:

```
yum install libevent-dev
```

После инсталляции требуемой библиотеки нам нужно выполнить конфигурационный сценарий Memcached, который создаст базу данных и некоторые таблицы в рамках конфигурирования Memcached. Этот конфигурационный сценарий уже имеется в наличии и находится в следующем местоположении:

```
<MYSQL_HOME>/share/innodb_memcached_config.sql
```

Здесь `<MYSQL_HOME>` – это каталог, где был установлен MySQL. Для Linux этот файл конфигурационного сценария по умолчанию будет находиться в `/usr/share/mysql/innodb_memcached_config.sql`.

Чтобы выполнить этот сценарий, надо в командной строке MySQL исполнить следующую ниже команду:

```
mysql> source <MYSQL_HOME>/share/innodb_memcached_config.sql
```

Приведенный выше сценарий создаст новую базу данных как `innodb_memcache`, которая будет иметь следующие таблицы:

- `containers`: это самая важная таблица базы данных Memcached. Эта таблица содержит все записи таблицы InnoDB, которые позволяют вам хранить значения Memcached. Для использования плагина Memcached таблица InnoDB должна зарегистрироваться в этой таблице. Эта таблица также описывает увязку со столбцами таблицы InnoDB;
- `cache_policies`: в этой таблице определяется политика кеша. В рамках одной политики она может иметь различные политики для операций `get`, `set`, `flush` и `delete`. В качестве значений политики относительно различных операций мы можем определить innodb_only (только InnoDB), cache-only (только кеш), caching (кеширование) и disable (дективировать);
- `config_options`: эта таблица имеет различные параметры Memcached, которые могут обновляться во время выполнения с использованием SQL. Например, мы можем настроить разделитель для идентификации различных значений столбцов из одного строкового значения.

Помимо этих баз данных и таблиц, Memcached также создаст тестовую базу данных, которая имеет только одну таблицу с именем `demo_test`. Эта таблица также увязана в таблице контейнера путем вставки записей для табличных данных.

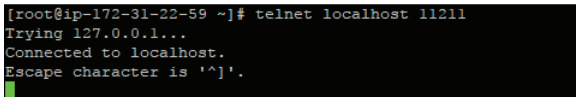
Теперь самое время активировать плагин `daemon_memcached`, выполнив следующую ниже команду в командной строке MySQL:

```
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

После успешной активации плагина Memcached он будет интегрирован в MySQL и будет автоматически активироваться всякий раз, когда сервер MySQL перезапускается.

Верификация

Как только вы активировали плагин Memcached, он начнет по умолчанию слушать порт 11211. Как показано на следующем ниже снимке экрана, к интерфейсу Memcached можно обращаться с помощью сеанса `telnet`:



Если вы успешно подключились через команду `telnet`, это будет означать, что Memcached был успешно интегрирован с MySQL. На интерфейсе Memcached можно выполнять определенные операции, связанные с хранением, получением или удалением данных.

Ниже приведен перечень основных команд плагина Memcached:

Команда	Описание
`get`	Извлечь данные по ключу
`set`	Сохранить пару ключ-значение в Memcached. Она также обновит значение, в случае если ключ уже существует, следовательно, это комбинация вставки и обновления
`add`	Эта команда вставит новый ключ-значение, и она не сработает, если ключ уже будет существовать
`replace`	Эта команда используется для обновления данных существующего ключа
`append`	Эта команда добавит значение в конец существующего значения, связанного с ключом
`prepend`	Эта команда добавит значение в начало существующего значения, связанного с ключом
`delete`	Удалить данные по их ключу

Использование плагина Memcached

Memcached можно использовать на различных этапах в реальных приложениях. Его интеграция с MySQL делает удобнее работу с большими данными. Давайте посмотрим на важность использования Memcached, разобравшись в различных его преимуществах.

Наладчик производительности

Работая с большими данными, нас всегда в первую очередь интересует производительность. Прежде чем приступать к работе, у нас может быть много вопросов, например как MySQL может быть оптимизирован с точки зрения производительности. Мы предполагаем, что управление большими данными с помощью MySQL может повлиять на производительность, так как это чистая структурированная база данных. Отнюдь нет! MySQL имеет довольно простую возможность работы с большими данными путем интеграции Memcached и использования его в качестве базы данных NoSQL. Благодаря этому уменьшается запаздывание в производительности и обеспечивается эффективность при управлении большими данными с помощью MySQL.

Плагин Memcached является одним из наладчиков производительности для MySQL при работе с большими данными. Memcached удаляет слой SQL и обращается к таблицам подсистемы хранения данных InnoDB напрямую. Следовательно, такие служебные операции, как синтаксический анализ SQL, больше не будут выполняться, а это действительно повлияет на производительность. Memcached также будет обслуживать частые запросы данных из памяти, не заходя в базу данных, что ускоряет работу при извлечении данных.

Инструмент кеширования

Memcached может использоваться как инструмент кеширования, поскольку он кеширует данные в памяти. Буферный пул управляется плагином Memcached, который кеширует данные InnoDB и обслуживает запрос, не входящий в таблицу InnoDB. В принципе, нам не нужно загружать данные в кеш вручную, они автоматически сохраняются в кеш-памяти всякий раз, когда делается первый запрос из приложения. Memcached в MySQL также обеспечивает вам постоянство хранения данных прямо в памяти, чтобы вы могли их использовать содержательно для различных типов данных без их потери. Memcached тоже можно запускать и останавливать без потери обновлений, сделанных в кешированных данных. Эти передачи данных из памяти в таблицы InnoDB автоматически обрабатываются плагином, не заставляя нас беспокоиться о согласованности данных.

Простота в использовании

Плагин Memcached довольно прост в использовании с реальными приложениями и позволяет легко работать с большими данными, обеспечивая большую производительность. Теперь СУБД MySQL 8 имеет плагин `daemon_memcached` в пределах самого сервера MySQL, что делает его легковесным и легким в управлении, поскольку он не потребляет никаких дополнительных ресурсов и ликвидирует накладные расходы управления отдельным сервером Memcached.

Анализ хранящихся в Memcached данных

Пришло время закатать рукава и поэкспериментировать с интерфейсом Memcached. Запись таблицы InnoDB должна быть в таблице контейнеров базы данных Memcached, готовая для использования этой таблицы через интерфейс Memcached. По умолчанию при установке плагина Memcached автоматически создается таблица `demo_test` в рамках тестовой базы данных `test`, и та же самая табличная запись вставляется в таблицу контейнеров. Определим новое отображение и выполним несколько операций. Давайте перейдем к следующему разделу!

Для хранения данных о результатах успеваемости студентов мы создали таблицу `student_result`, используя для этого следующий ниже запрос:

```
mysql> CREATE TABLE student_result (
student_id INT PRIMARY KEY,
student_name VARCHAR(20),
maths INT,
english INT,
history INT,
flags INT,
cas BIGINT UNSIGNED,
expiry INT);
```

Увязку таблицы `student_result` можно выполнить, вставив запись в таблицу `containers` следующим образом:

```
mysql> INSERT INTO innodb_memcache.containers
(name, db_schema, db_table, key_columns, value_columns, flags, cas_column,
expire_time_column, unique_idx_name_on_key)
VALUES ('default', 'test', 'student_result', 'student_id',
'student_name,maths,english,history','flags','cas','expiry', 'PRIMARY');
```

Чтобы использовать это новое сопоставление, рекомендуется перезапустить Memcached с помощью следующих ниже команд:

```
mysql> UNINSTALL PLUGIN daemon_memcached;
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

Теперь все готово для того, чтобы выполнить несколько запросов через Memcached. Давайте обратимся к терминалу Memcached через `telnet`, чтобы сохранить и получить отметки студентов. Для этого надо сделать следующее:

1. Вставить новую запись:

```
telnet localhost 11211
 Trying 127.0.0.1...
 Connected to localhost.
 Escape character is '^]'.
 set 101 0 0 22
 Jaydip|50|60|70
 STORED
```

   После выполнения команды `set` с соответствующим ключом и данными она будет автоматически сохранена в таблице `student_result`, как показано на следующем ниже снимке экрана:

```
mysql> select * from test.student_result;
+------------+--------------+-------+---------+---------+-------+------+--------+
| student_id | student_name | maths | english | history | flags | cas  | expiry |
+------------+--------------+-------+---------+---------+-------+------+--------+
|        101 | Jaydip       |    50 |      60 |      70 |     0 |    2 |      0 |
+------------+--------------+-------+---------+---------+-------+------+--------+
1 row in set (0.00 sec)
```

2. Извлечь запись с ключом:

```
get 101
 VALUE 101 0 15
 Jaydip|50|60|70
```

Приведенная выше команда выберет только одну строку, но она также поддерживает получение нескольких пар ключ-значение в одном запросе, что повышает производительность за счет снижения коммуникационного трафика. Передача нескольких ключей с помощью команды `get` приведет к получению данных от каждого ключа. В следующем ниже примере показана поддержка нескольких команд `get`.

3. Извлечь многочисленные записи:

```
get 101 102
 VALUE 101 0 15
 Jaydip|50|60|70
 VALUE 102 0 14
 Keyur|45|63|87
 END
```

Плагин `daemon_memcached` также поддерживает диапазонные запросы, которые могут быть полезны при поиске записей с помощью различных операторов, таких как `<`, `>`, `<=` и `>=`. Поиск записи по условию значительно облегчается при помощи диапазонных запросов. Перед оператором должен находиться символ `@`.

4. Диапазонные запросы:

```
get @>101
 VALUE 102 0 14
 Keyur|45|63|87
 END
```

В результате будут выбраны все записи, которые больше ключа 101.

Конфигурирование репликации Memcached

Выражаясь простыми словами, репликация – это копирование данных с одного сервера на один или несколько серверов. Репликация позволяет распределить нагрузку на несколько серверов, что повышает производительность и снижает вероятность повреждения данных, так как данные распределяются между различными серверами. MySQL 8 поддерживает различные методы репликации, но традиционный метод – реплицировать через двоичные журналы ведущего сервера. Двоичный журнал – это файл, который собирает все запросы, пытающиеся изменить табличные данные. Эти запросы будут выполнены на ведомых серверах для репликации данных на ведущем сервере.

Плагин daemon_memcached имеет возможность управлять двоичным журналом на ведущем сервере, и то же самое может использоваться для репликации на ведомых серверах. Все команды Memcached поддерживаются двоичным журналированием.

Для включения репликации двоичных журналов в MySQL 8 выполните следующие ниже шаги.

1. Во время запуска сервера добавьте системные переменные --innodb_api_enable_binlog=1 и --log-bin, чтобы активировать двоичный журнал на ведущем сервере:

```
mysqld ... --log-bin --innodb_api_enable_binlog=1
```

2. Установите ведущий сервер репликации, добавив в файл my.cnf или my.ini следующие параметры. Идентификатор сервера server-id должен быть уникальным для каждого сервера. Если вы опустите идентификатор сервера (или зададите ему явным образом значение по умолчанию 0), ведущий сервер откажется от любых соединений с ведомыми серверами. После добавления следующих ниже свойств сервер MySQL требует перезапуска:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

3. Установите ведомый сервер репликации, добавив в файл my.cnf или my.ini следующие свойства. Остановите работу сервера и отредактируйте файл, добавив уникальный идентификатор сервера для каждого ведомого сервера:

```
[mysqld]
server-id=2
```

4. Все серверы должны быть синхронизированы; следовательно, возьмите резервную копию ведущей базы данных и восстановите ее во всех ведомых серверах.
5. Исполните следующую ниже команду, чтобы получить координаты двоичных журналов ведущего сервера:

```
mysql> SHOW MASTER STATUS;
```

6. Выполните следующую ниже команду на ведомом сервере для настройки ведомого сервера с помощью двоичных журналов ведущего сервера:

```
mysql> CHANGE MASTER TO
MASTER_HOST='localhost',
MASTER_USER='root',
MASTER_PASSWORD='',
MASTER_PORT = 13000,
MASTER_LOG_FILE='0.000001,
MASTER_LOG_POS=114;
```

7. Исполните следующую ниже команду, чтобы запустить ведомый сервер:

```
mysql> START SLAVE;
```

Теперь все необходимые настройки готовы для репликации с помощью Memcached. Мы можем верифицировать конфигурацию, исполнив запросы через Memcached, и те же самые данные распространятся в ведущем и ведомых серверах.

API Memcached для различных технологий

Приложение может хранить и извлекать данные через интерфейс Memcached, используя различные доступные API. Прикладные интерфейсы Memcached имеются на разных языках программирования, таких как Perl, Python, Java, PHP, C и Ruby. С помощью API Memcached приложение может взаимодействовать с интерфейсом Memcached для операций хранения и извлечения информации.

Memcached хранит информацию в кеше, на который в качестве ключа может ссылаться одно строковое значение. Этот ключ используется в качестве ссылки для извлечения и хранения данных. Memcached всегда обслуживает запросы данных на основе ключа, выступающего как ссылка. Вследствие этого кеш работает как большой ассоциативный массив или хеш-таблица.

Интерфейс Memcached имеет универсальный метод хранения и извлечения информации на всех языках программирования, хотя механизм для конкретного языка может отличаться. Рассмотрим несколько распространенных методов и их использование:

- `get(ключ)`: этот метод используется для извлечения информации из кеша относительно предоставленного ключа. Он возвращает данные, связанные с ключом, или же возвращает значение null;
- `set(ключ, значение [, время_истечения])`: этот метод используется для хранения информации в кеше с указанным ключом. Он автоматически обновит данные в случае, если тот же самый ключ уже существует. В противном случае он создаст в кеш-памяти новую пару ключ-значение. Третий аргумент в этом методе является необязательным; в нем можно задать время истечения данных. При указании времени истечения он будет автоматически удалять кешированные данные, когда текущее время совпадает со временем истечения. Это время истечения должно быть в секундах;
- `add(ключ, значение [, время_истечения])`: этот метод добавляет данные со ссылкой на ключ, только если заданный ключ еще не существует. Он похож на метод `set` с той лишь разницей, что он не будет обновлять данные, если ключ уже существует в кеше. Мы можем указать время истечения в секундах, чтобы очистить данные после установленного времени;
- `replace(ключ, значение [, время_истечения])`: этот метод обновляет существующие данные новыми данными, только если ключ в Memcached уже существует;
- `delete(ключ [, время])`: этот метод будет удалять данные, связанные с предоставленным ключом. Мы можем также задать время как необязательный аргумент, для того чтобы заблокировать добавление нового элемента на обусловленный период с таким же ключом;
- `incr(ключ, значение)`: этот метод увеличит существующие данные на заданное значение связанного ключа;
- `decr(ключ, значение)`: этот метод уменьшит существующие данные на заданное значение связанного ключа;
- `flush_all`: этот метод приведет к аннулированию всех элементов в кеше. Он не будет их удалять, но они молча будут удалены из кеша.

Memcached с Java

Приложения Java могут быть легко интегрированы с интерфейсом Memcached с помощью клиента, который уже имеется в Java. Этот клиент Java доступен в ре-

позитории GitHub, и вы можете его скачать с https://github.com/gwhalin/Memcached-Java-Client/downloads.

Все классы Java в рамках пакета `com.danga.MemCached` предоставляют родной интерфейс для экземпляров Memcached. Предпочтительно использовать форматы сериализации JSON или недвоичной сериализации данных вместо функционала сериализации Java.

Вы также можете получить клиента Memcached для своего maven-проекта со следующей зависимостью:

```
<dependency>
 <groupId>com.whalin</groupId>
 <artifactId>Memcached-Java-Client</artifactId>
 <version>3.0.2</version>
</dependency>
```

Давайте посмотрим, как подключиться к экземплярам Memcached, используя интерфейс Java.

1. Создать экземпляр `MemCachedClient`:

   ```
   MemCachedClient memCachedClient = new MemCachedClient();
   ```

2. Сконфигурировать список серверов и весов с помощью `SockIOPool`:

   ```
   SockIOPool pool = SockIOPool.getInstance();
    pool.setServers( servers );
    pool.setWeights( weights );
   ```

   Здесь `servers` содержат массив экземпляров Memcached, которые могут использоваться, а `weights` – это отдельный вес каждого сервера.

3. В `SockIOPool` также можно задать различные свойства подключения, такие как минимальное подключение, начальное подключение, время простоя и т. д. Эти свойства являются необязательными и зависят от потребностей разработчика.

   ```
   pool.setInitConn( 5 );
   pool.setMinConn( 5 );
   pool.setMaxConn( 250 );
   ```

4. Инициализируйте пул `SockIOPool`, после того как вы закончите с конфигурацией параметров подключения:

   ```
   pool.initialize();
   ```

Давайте посмотрим полный пример, который описывает интеграцию с клиентом Memcached:

```
class MemcachedUtil {
private static String[] servers = {"localhost:11211"};
private static Integer[] weights = { 1 };
public MemCachedClient getMemCachedClient() {
SockIOPool pool = SockIOPool.getInstance();
pool.setServers(servers);
pool.setWeights( weights );
 if (!pool.isInitialized()) {
 pool.setInitConn( 5 );pool.setMinConn( 5 );pool.setMaxConn( 250
);pool.initialize(); }MemCachedClient memCachedClient = new
```

```
MemCachedClient();memCachedClient.setCompressEnable(false);
memCachedClient.setCompressThreshold(0);
 return memCachedClient;
}
}
```

Memcached с PHP

Интерфейс Memcached также может быть интегрирован с приложениями PHP через расширение PECL. Следующая ниже команда поможет вам установить расширение PECL для систем на базе Linux:

```
yum --install php-pecl-memcache
```

Можно также использовать следующую ниже команду:

```
apt-get install php-memcache
```

После его установки мы можем задать глобальные параметры конфигурации в файле `php.ini`.

Давайте посмотрим, как подключить Memcached из приложения PHP.

Создайте объект Memcached, подключившись к серверам следующим образом:

```
$memcache = new Memcache;
$memcache->addServer('localhost',11211);
```

Мы можем добавить несколько серверов, вызвав метод `addServer`, указав имя хоста и номер порта в качестве аргументов. Это не сразу откроет соединение между интерфейсом Memcached и приложением: оно откроется только после того, как вы попытаетесь сохранить или получить значение.

Давайте взглянем на один небольшой пример получения данных Memcached:

```
<?php
 $memcache = new Memcache;
 $memcache->addServer('10.12.4.15',11211);
 $memcache->addServer('10.12.4.16',11211);
 $memcache->set('maths','60');
 echo $memcache->get('maths');
?>
```

Memcached с Ruby

Для интеграции Memcached с языком Ruby существует целый ряд модулей. Клиент `Ruby-MemCache` является одним из них. Он предоставляет интерфейс для Memcached. Мы можем установить клиент `Ruby-MemCache`, используя `RubyGems`, с помощью следующей ниже команды:

```
gem install Ruby-MemCache
```

Эта команда автоматически установит клиент и все необходимые для него библиотеки. После установки этого клиента мы можем обратиться к интерфейсу Memcached следующим образом:

1. Создать клиента Memcached:

```
require 'memcache'
memc = MemCache::new ' 10.12.4.15:11211:1'
```

Здесь единица после номера порта является весом конкретного сервера. Мы также можем добавить несколько серверов, добавив к тому же самому объекту Memcached, следующим образом:

```
memc += ["10.12.4.16:11211:2"]
```

2. Установить данные в Memcached, выполнив присвоение:

```
memc["key"] = "value"
```

3. Получить данные из Memcached:

```
print memc["key"]
```

Memcached с Python

Для доступа к интерфейсу Memcached с помощью приложения Python необходимо установить модуль Python Memcached, который можно скачать с https://www.tummy.com/software/python-memcached/.

После скачивания указанного пакета его можно установить с помощью установщика Python следующим образом:

```
python setup.py install
```

После установки этого модуля к нему очень легко обратиться, создав новый экземпляр класса `memcache.Client`:

```
import memcache
memcached = memcache.Client(['localhost:11211'])
```

1. Установить данные в Memcached:

```
memcached.set('key','value')
```

2. Получить данные из Memcached:

```
memc.get('key')
```

При доступе к интерфейсу Memcached все языки программирования ведут себя почти одинаково. В интерфейсах для всех языков также доступны универсальные функции, такие как `get`, `set`, `add`, `replace`, `delete` и т. д. Подавляющая часть функциональности остается одинаковой на всех языках, и разница лишь в синтаксисе, используемом в разных языках программирования.

Резюме

В этой главе мы обсудили использование плагина Memcached с MySQL 8 для применения с большими данными и архитектуру всей системы, в которой Memcached занимает видное место. Вы узнали, как Memcached может быть установлен в системе на базе Linux, и мы также обсудили использование Memcached в реальных приложениях. Кроме того, мы также поэкспериментировали с данными, используя интерфейс Memcached, и различными методами запроса. Мы рассмотрели конфигурацию Memcached, которая должна быть выполнена для целей репликации. Наконец, мы увидели различные API, доступные для различных языков программирования, включая Java, PHP, Ruby и Python.

В следующей главе мы увидим, каким образом при помощи различных методов разделения крупные данные могут быть разделены, чтобы обеспечить легкое управление большими данными в MySQL 8.

Глава 5

Разделение больших объемов данных

В предыдущей главе вы познакомились с основами Memcached, программным обеспечением, реализующим службу кеширования данных в оперативной памяти на основе хеш-таблицы, и его замечательными особенностями. Мы также подробно рассказали о его работе в различных сценариях и о том, как мы можем использовать API Memcached для повышения производительности нашего запроса в MySQL. В этой главе давайте теперь рассмотрим различные методы разделения и как разделение способно помочь в случае крупных таблиц данных.

В данной главе мы рассмотрим следующие ниже темы, связанные с разделением данных в MySQL 8:

- разделение данных в MySQL 8;
- горизонтальное разделение MySQL 8;
- вертикальное разделение;
- подрезка разделения в MySQL;
- выполнение запросов на разделенных данных.

Когда мы говорим о больших данных, автоматически считается, что нам придется работать с крупными таблицами данных, где хранится различная информация. Для любой организации очень важно хранить данные таким образом, чтобы СУБД обеспечивала масштабируемость, производительность, доступность и безопасность. Например, в высокодоступном магазине электронной коммерции часто размещаются тысячи заказов или более. Поэтому, чтобы поддерживать дневную доставку заказов, показывая панель мониторинга текущих заказов, требуется запрашивать таблицу, показывающую заказы за последние пять лет; выполнение этого процесса с текущими данными займет много времени. Здесь исторические данные о заказах необходимы для аналитической цели – чтобы выяснить поведение пользователей или тенденции, но при этом требуется выполнять такой анализ на ограниченных наборах данных.

Существуют различные способы достижения наилучшего подходящего решения для обеспечения высокой доступности, масштабируемости и высокопроизводительной архитектуры; ключевым компонентом является разделение данных. В базе данных данные в каждой таблице хранятся в физических файловых группах. Поэтому, разделяя эту таблицу данных из одной файловой группы на несколько файловых групп, можно сократить размер файлов и создать масштабируемую и высокопроизводительную базу данных.

Ниже перечислены ключевые преимущества использования разделения в базе данных.

- **Масштабируемость**: поскольку данные будут совместно использоваться более чем одним разделом, серверы могут быть сконфигурированы для использования многочисленных узлов, а разделы могут быть сконфигурированы между многочисленными узлами. Это позволит устранить любые аппаратные ограничения и даст возможность в значительной степени масштабировать базу данных для размещения крупных объемов данных.
- **Высокая производительность**: поскольку данные хранятся в многочисленных разделах, каждый запрос будет выполняться на небольшой части данных. Например, чтобы в магазине электронной коммерции с историей заказов более двух лет получить список заказов, размещенных в текущем месяце, потребуется проверить только один раздел, а не всю историю заказов, тем самым сократив время выполнения запроса. Чтобы получить запрос на более чем одном разделе, мы также можем запустить его параллельно, тем самым уменьшив общее время для извлечения данных из базы данных.
- **Высокая доступность**: при разделении данные делятся на несколько файловых групп. Каждая файловая группа логически связана, но может быть доступна и работать независимо. Таким образом, если одна из файловых групп или один из разделов будут повреждены или один из узлов на сервере аварийно завершит свою работу, то мы не потеряем доступа ко всей таблице, а только часть базы данных будет недоступна, что исключит вероятность аварийного завершения работы всей системы и сделает систему высокодоступной.
- **Безопасность**: может оказаться так, что некоторые данные в таблицах требуют принятия мер повышения уровня безопасности от кражи или утечки данных. Разделение данных позволяет обеспечить дополнительную безопасность для одного или нескольких разделов, чтобы избежать проблем с безопасностью, тем самым повышая доступность данных.

Разделение данных в MySQL 8

В общих чертах разделение используется для разбиения общего целого на несколько логически связанных подгрупп, так чтобы каждая подгруппа могла быть идентифицирована независимо и могла быть скомбинирована в единый раздел. Давайте разберемся, что такое разделение в терминах **реляционных СУБД**.

Что такое разделение данных?

Разделение обычно используется для разбиения данных на несколько логических файловых групп с целью повышения производительности, доступности и управляемости. Во время работы с большими данными стандартной является ситуация, когда данные насчитывают миллиарды записей. Поэтому для повышения производительности базы данных лучше всего разделять данные между несколькими файловыми группами. Эти файловые группы могут быть на одной машине или на нескольких машинах и идентифицироваться по ключу. Эти файловые группы называются **разделенными данными**.

Типы разделения данных

Данные в таблице могут быть разделены двумя способами:

- горизонтальным разделением;
- вертикальным разделением.

Горизонтальное разделение

Если число строк в таблице очень велико, таблицу можно разделить на многочисленные разделы в качестве так называемого **горизонтального разделения**. Когда используется горизонтальное разделение, каждый раздел таблицы содержит одинаковое количество столбцов. Можно получить доступ ко всем разделам сразу, либо можно получить доступ к каждому разделу по отдельности.

Вертикальное разделение

При вертикальном разделении столбцы таблиц разделяются для повышения производительности и улучшения управления базой данных. Вертикальное разделение может быть достигнуто двумя способами. Первый – это нормализация таблиц. Вместо того чтобы в таблице иметь слишком много столбцов, столбцы могут быть разделены на несколько таблиц путем деления данных. Второй – создание для определенных столбцов в таблице отдельных физических файловых групп. Вертикальное разделение в настоящее время в MySQL 8 не поддерживается.

В MySQL при разделении все разделы таблицы должны использовать одну и ту же подсистему хранения данных. В MySQL 8 разделение поддерживается только в подсистеме хранения данных InnoDB. Поэтому все разделы таблицы и всех таблиц, которые, возможно, должны быть разделены, должны использовать InnoDB в качестве подсистемы хранения данных. В MySQL 5.7 разделение также поддерживалось в подсистеме хранения данных NDB Cluster. Однако на данный момент в MySQL 8 разделение в кластерной подсистеме NDB Cluster не поддерживается, поэтому InnoDB остается единственной подсистемой с поддержкой разделения, которая широко используется.

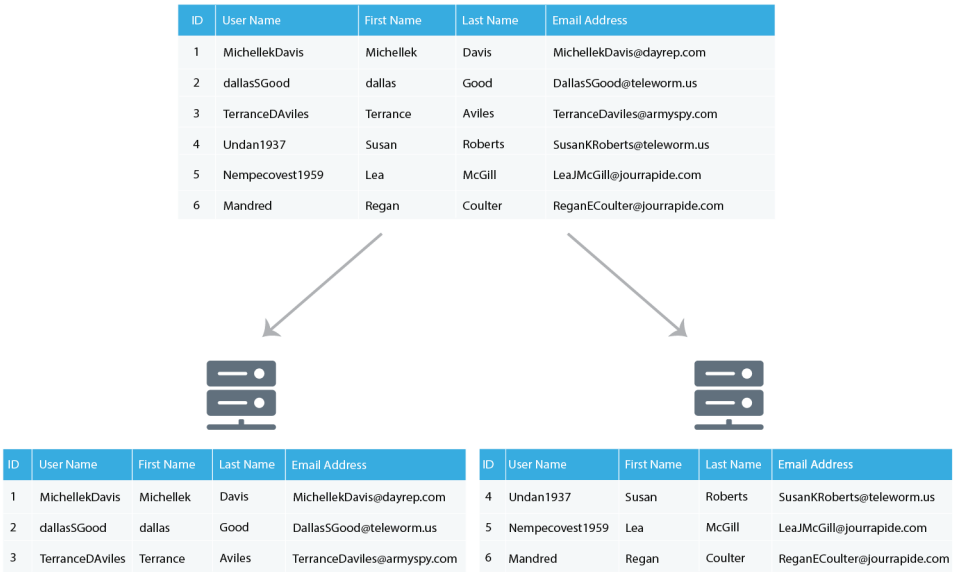
Давайте рассмотрим некоторые преимущества, связанные с разделением данных.

- Если таблица содержит исторические данные, такие как журналы приложения, данные старше шести месяцев не имеют для активной работы приложения никакого значения. Если разделение создается на основе месяцев, то один из разделов можно легко удалить.
- В том же самом случае с журналами, если мы хотим фильтровать данные между двумя датами, оптимизатор MySQL может идентифицировать конкретные разделы, где он может найти отфильтрованные записи, что в конечном итоге может привести к гораздо более быстрым результатам запроса, поскольку число сканируемых строк резко сокращается.
- MySQL 8 также поддерживает запрос данных на определенных разделах. Это может сократить количество проверяемых записей, когда вы знаете раздел, который должен быть запрошен для требуемых данных.

Горизонтальное разделение в MySQL 8

Как уже отмечалось, горизонтальное разделение создает более одного раздела данных. Каждый раздел будет иметь одинаковое количество столбцов. Для деле-

ния страницы на несколько разделов используются различные методы разделения. Рассмотрим следующий ниже рисунок горизонтального разделения:



ID	User Name	First Name	Last Name	Email Address
1	MichellekDavis	Michellek	Davis	MichellekDavis@dayrep.com
2	dallasSGood	dallas	Good	DallasSGood@teleworm.us
3	TerranceDAviles	Terrance	Aviles	TerranceDaviles@armyspy.com
4	Undan1937	Susan	Roberts	SusanKRoberts@teleworm.us
5	Nempecovest1959	Lea	McGill	LeaJMcGill@jourrapide.com
6	Mandred	Regan	Coulter	ReganECoulter@jourrapide.com

Ниже представлен список типов разделения, поддерживаемых в MySQL 8:

- диапазонное разделение;
- списковое разделение;
- хеш-разделение;
- столбцовое разделение;
- разделение по ключу;
- разбиение на подразделы.

Давайте подробно разберемся в каждом типе разделения.

Диапазонное разделение

Когда разделение выполняется на основе выражений, содержащих непрерывные неповторяющиеся значения диапазонов, оно называется **диапазонным разделением**. Выражение для диапазонного разделения содержит ключевой оператор `VALUE LESS THAN`.

Существует несколько типов данных, на основе которых мы можем разделить таблицу:

```
CREATE TABLE access_log (
 log_id INT NOT NULL,
 type VARCHAR(100),
 access_url VARCHAR(100),
 access_date TIMESTAMP NOT NULL,
 response_time INT NOT NULL,
 access_by INT NOT NULL
)
PARTITION BY RANGE (UNIX_TIMESTAMP(access_date)) (
```

```
 PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2017-05-01 00:00:00')),
 PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2017-09-01 00:00:00')),
 PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2018-01-01 00:00:00')),
 PARTITION p3 VALUES LESS THAN (UNIX_TIMESTAMP('2018-05-01 00:00:00')),
 PARTITION p4 VALUES LESS THAN (UNIX_TIMESTAMP('2018-09-01 00:00:00')),
 PARTITION p5 VALUES LESS THAN (UNIX_TIMESTAMP('2019-01-01 00:00:00')),
);
```

Таким образом, согласно предыдущему запросу, для таблицы `access_log` с журналом регистрации доступа будет создано шесть разделов. Согласно разделу, каждый раздел будет хранить данные о доступе за четыре месяца. Так, скажем, когда новый журнал регистрации доступа придется на дату 15 августа, он автоматически перейдет в раздел `p2`. Что произойдет, если какие-либо данные появятся после 1 января 2019 года? Поскольку для любой даты после 2018 года диапазон не определен, при попытке вставить данные за пределами диапазона будет возвращена ошибка. Поэтому для преодоления такой ситуации мы должны задать верхнее значение диапазона. В MySQL 8 мы можем предоставить верхний предел, используя ключевое слово `MAXVALUE`. Ключевое слово `MAXVALUE` всегда будет принимать максимально возможное значение для столбцов, основываясь на типе данных. Для этого можно выполнить следующий ниже запрос на разделение:

```
PARTITION BY RANGE (UNIX_TIMESTAMP(access_date)) (
 PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2017-05-01 00:00:00')),
 PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2017-09-01 00:00:00')),
 PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2018-01-01 00:00:00')),
 PARTITION p3 VALUES LESS THAN (UNIX_TIMESTAMP('2018-05-01 00:00:00')),
 PARTITION p4 VALUES LESS THAN (UNIX_TIMESTAMP('2018-09-01 00:00:00')),
 PARTITION p5 VALUES LESS THAN (UNIX_TIMESTAMP('2019-01-01 00:00:00')),
 PARTITION p6 VALUES LESS THAN MAXVALUE,
);
```

Мы создали еще один раздел, `p6`, с максимальным значением. Итак, теперь, если какой-либо журнал регистрации доступа появится 1 января 2019 года или после этой даты, он перейдет в раздел `p6`.

Это позволит нам упростить выполнение двух задач:

- теперь это даст нам возможность удалять устаревшие и бесполезные данные. Например, мы можем удалить раздел `p0` в 2018 году, поскольку он будет иметь меньшее влияние на данные;
- в любой момент времени мы можем изменить критерии для раздела с помощью запроса на изменение таблицы `ALTER`. Поэтому, если мы хотим воссоздать раздел после 2018 года, это будет простой задачей.

При использовании диапазонного разделения со столбцами `TIMESTAMP` выражение `UNIX_TIMESTAMP` является единственным, которое можно использовать для разделения данных. При применении выражений в диапазонном разбиении оно должно возвращать значение, которое может использоваться выражением `VALUE LESS THAN`, иначе MySQL выдаст ошибку.

Если при вставке данных в базу данных значение столбца, используемого для разделения, равно *null*, то MySQL добавит вновь вставляемую строку данных в самый младший раздел.

Списковое разделение

В списковом разделении разделяющее выражение содержит синтаксическую конструкцию, которая сверяется со списком значений, и разделы создаются, опираясь на совпадающие значения. Выражения для спискового разбиения содержат ключевой оператор `VALUES IN`:

```
CREATE TABLE access_log (
 log_id INT NOT NULL,
 type VARCHAR(100),
 access_url VARCHAR(100),
 access_date TIMESTAMP NOT NULL,
 response_time INT NOT NULL,
 access_by INT NOT NULL
 website_id INT
)
PARTITION BY LIST(website_id) (
 PARTITION PartitionNorth VALUES IN (1,2),
 PARTITION PartitionSouth VALUES IN (3,4),
 PARTITION PartitionWest VALUES IN (5,6),
 PARTITION PartitionEast VALUES IN (7,8)
);
```

Если журнал регистрации доступа принадлежит нескольким веб-сайтам, то мы можем разделить таблицу журнала регистрации на многочисленные разделы. Здесь мы разделили данные журнала регистрации по их регионам. Все журналы регистрации с веб-сайтов, принадлежащих северному региону страны, будут находиться в одном разделе, что может быть полезно для анализа данных журнала регистрации на основе региона страны.

В списковом разделении значения `NULL` в ключах с определением раздела принимаются только в том случае, если любой из списков значений разделов содержит значение `NULL` в качестве допустимого значения. Если мы попытаемся вставить любую запись, имеющую значение `NULL`, в ключ, используемый для раздела, и `NULL` не является допустимым значением для любого из разделов, то это вызовет ошибку.

С учетом предыдущего примера рассмотрим следующий ниже список разделов:

```
PARTITION BY LIST(website_id) (
 PARTITION PartitionNorth VALUES IN (1,2),
 PARTITION PartitionSouth VALUES IN (3,4),
 PARTITION PartitionWest VALUES IN (5,6),
 PARTITION PartitionEast VALUES IN (7,8)
 PARTITION PartitionNoRegion Values IN (NULL)
);
```

Здесь мы определили новый раздел, раздел без области `PartitionNoRegion`. Если в журнале регистрации доступа мы не сможем идентифицировать регион страны и если будет иметься шанс, что идентификатор веб-сайта `website_id` будет иметь значение `NULL`, то эти данные будут сохранены в новом созданном разделе.

Хеш-разделение

В хеш-разделении для идентификации разделов используются пользовательские выражения. В хеш-разделении количество разделов предопределено, и вставля-

емые данные разделяются, опираясь на выражения, применяемые к значениям столбцов. Хеш-разделение распределяет данные равномерно. Требуется, чтобы выражение возвращало неотрицательные целочисленные значения:

```
CREATE TABLE access_log (
 log_id INT NOT NULL,
 type VARCHAR(100),
 access_url VARCHAR(100),
 access_date TIMESTAMP NOT NULL,
 response_time INT NOT NULL,
 access_by INT NOT NULL
 website_id INT
)
PARTITION BY HASH(website_id)
PARTITIONS 4;
```

Как показано в предыдущем запросе, нам не нужно явно определять, в какой раздел следует вставить значение. Единственное, что мы здесь определили, – это столбец хеширования, идентификатор веб-сайта `website_id` и количество разделов для таблицы. Количество разделов является необязательным, и если это число не определено, то по умолчанию для таблицы будет использоваться один раздел.

В хеш-разделении номер раздела определяется с помощью функции деления по модулю:

```
номер раздела = Остаток(выражение, количество разделов)
```

В предыдущем примере, если новое вставляемое значение имеет идентификатор веб-сайта 2:

```
номер раздела = Остаток(2,4)
номер раздела = 2
```

Таким образом, результирующие данные будут храниться в разделе номер 2. Номер раздела идентифицирован на основе результата предыдущей формулы.

Если в хеш-разделении `NULL` используется в качестве значения в столбце с определением раздела, то такое значение рассматривается как 0.

Линейное хеш-разделение представляет собой вариант хеш-разделения. Там, где в обычном хеш-разделении для идентификации номера раздела используется деление по модулю, в линейном хеш-разделении для идентификации номера раздела используется два алгоритма:

```
CREATE TABLE access_log (
 log_id INT NOT NULL,
 type VARCHAR(100),
 access_url VARCHAR(100),
 access_date TIMESTAMP NOT NULL,
 response_time INT NOT NULL,
 access_by INT NOT NULL
 website_id INT
)
PARTITION BY LINEAR HASH(website_id)
PARTITIONS 4;
```

Для линейного хеширования вычисление номера раздела выполняется по следующей формуле:

```
результирующее значение = Степень(2, CEILING(LOG(2, количество разделов))),
```

Предыдущее выражение отыщет значение, опираясь на количество разделов, заданных в определении CREATE TABLE:

```
номер раздела = (выражение) & (результирующее значение)
```

Чтобы выяснить номер раздела, MySQL выполнит побитовую операцию & на результирующем значении, полученном из выражения один. Когда номер раздела больше или равен количеству разделов, то MySQL воспользуется еще одним выражением, чтобы найти меньшее число:

```
While
 номер раздела >= количество разделов:
 результирующее значение = результирующее значение / 2
 номер раздела = номер раздела & (результирующее значение - 1)
```

Таким образом, MySQL будет в цикле повторять блок кода до тех пор, пока он не найдет номера раздела, который меньше количества разделов, определенных в таблице.

Когда используется линейное хеш-разделение, то удаление, разбиение, добавление или слияние разделов выполняется намного быстрее, по сравнению с обычным хеш-разделением. Однако, по сравнению с хеш-разделением, данные не распределяются должным образом.

Столбцовое разделение

Столбцовое разделение на самом деле является еще одним вариантом диапазонного разделения и спискового разделения. Столбцовое разделение в выражении разделения поддерживает строковые столбцы, такие как varchar, char, binary и varbinary. Рассмотрим подробно оба типа столбцового разделения.

Диапазонное столбцовое разделение

Столбцовое разделение отличается от диапазонного разделения. Диапазонное столбцовое разделение поддерживает в выражении только имя столбцов. Кроме того, для диапазонного разделения может использоваться более одного столбца. В качестве входных данных для диапазонного столбцового разделения можно использовать столбцы с типами date, datetime или string.

```
CREATE TABLE имя_таблицы
 PARTITION BY RANGE COLUMNS(список_столбцов) (
 PARTITION имя_раздела VALUES LESS THAN (список_значений)[,
 PARTITION имя_раздела VALUES LESS THAN (список_значений)][, ...]
)
```

Как определено в предыдущем фрагменте кода, можно определить список столбцов, на основе которого можно создать разделение таблицы:

```
список_столбцов: имя_столбца[, имя_столбца][, ...]
```

В списке значений и в списке столбцов должно быть равное количество полей:

```
список_значений: значение[, значение][, ...]
```

Давайте разберемся в этом равенстве на примере:

```
CREATE TABLE access_log (
 log_id INT NOT NULL,
 type VARCHAR(100),
 access_url VARCHAR(100),
 access_date TIMESTAMP NOT NULL,
 response_time INT NOT NULL,
 access_by INT NOT NULL
 website_id INT
)
PARTITION BY RANGE COLUMNS(website_id, log_id)
PARTITION partition0 VALUE LESS THAN (1,10000)
PARTITION partition1 VALUE LESS THAN (2,10000)
PARTITION partition2 VALUE LESS THAN (1,100000)
PARTITION partition3 VALUE LESS THAN (2,100000)
PARTITION partition4 value LESS THAN (MAXVALUE, MAXVALUE)
```

Здесь, как вы видите, создано пять разделов, которые хранят строки, разделенные по идентификатору веб-сайта `website_id` и идентификатору журнала `log_id`. При диапазонном столбцовом разделении раздел создается на основе сравнения сочетания столбцов, а не отдельных значений, как было в случае диапазонных разделов. Как показано в примере, `partition2` имеет значение идентификатора веб-сайта 1, которое меньше, чем у `partition1`, имеющего идентификатор веб-сайта 2. Однако это является допустимым, поскольку сочетание сайта и идентификатора журнала `(1,100000)` больше `(2, 10000)`.

В предыдущем примере мы применили `MAXVALUE` в качестве последнего раздела, однако можно иметь более одного раздела с `MAXVALUE` в качестве списка для одного из столбцов, указанных в выражении раздела.

Рассмотрим следующий ниже пример, чтобы разобраться в `MAXVALUE` лучше:

```
PARTITION BY RANGE COLUMNS(website_id,log_id)
PARTITION partition0 VALUE LESS THAN (1,10000)
PARTITION partition1 VALUE LESS THAN (MAXVALUE,100000)
PARTITION partition2 VALUE LESS THAN (MAXVALUE,1000000)
PARTITION partition3 value LESS THAN (MAXVALUE, MAXVALUE)
```

Как показано в предыдущем примере, идентификатор веб-сайта `website_id` столбца имеет максимальное значение в разделах 1, 2 и 3, в то время как второй параметр, идентификатор журнала `log_id` раздела, увеличивается линейно в каждом разделе, что является полностью допустимым сценарием.

Списковое столбцовое разделение

Как и диапазонное столбцовое разделение, списковое столбцовое разделение также отличается от спискового разделения. В списковом столбцовом разделении можно использовать многочисленные столбцы и допустимы значения столбцов с типом `string`, `datetime` и `date`.

Ниже приведена синтаксическая конструкция, используемая для создания спискового столбцового разделения:

```
CREATE TABLE имя_таблицы
PARTITION BY LIST COLUMNS(список_столбцов) (
PARTITION имя_раздела VALUES IN [(список_значений1), (список_значений2) ...]
```

Как показано в этой синтаксической конструкции, списковое столбцовое разделение может иметь многочисленные списки значений. По каждому столбцу будет указан список значений в разделе, который должен использоваться для вставки строк в раздел:

```
CREATE TABLE employee (
 first_name VARCHAR(25),
 last_name VARCHAR(25),
 joining_year INT,
 designation VARCHAR(100),
)
PARTITION BY LIST COLUMNS(joining_year,designation) (
 PARTITION pRegion_1 VALUES IN((2011,2012),('jr. consultant','sr.consultant')),
 PARTITION pRegion_2 VALUES IN((2013,2014), ('team lead', 'system architect')),
 PARTITION pRegion_3 VALUES IN((2014,2015), ('ui developer', 'ux developer')),
 PARTITION pRegion_4 VALUES IN((2016), ('project manager'))
);
```

Как показано здесь, для каждого раздела имеется два списка. Поэтому когда вставляется новая строка, то в зависимости от значения соответствующего разделения она перейдет в свой нужный раздел. Как и в диапазонном столбцовом разделении, столбец в нескольких разделах может иметь одинаковые значения, но ни один из двух разделов не может иметь одинаковую комбинацию значений.

Разделение по ключу

Разделение по ключу очень похоже на хеш-разделение. Хеш-разделение позволяет в рамках выражения раздела использовать функцию, в то время как разделение по ключу использует для разделения ключ таблицы.

Выражение для разделения по ключу использует ключевое слово `PARTITION BY KEY()`. Здесь ключ в качестве аргумента может принимать имя столбца. Если в ключе столбец не определен, то MySQL в качестве аргумента по умолчанию будет использовать первичный ключ. В случае отсутствия первичного ключа в таблице MySQL для определения разделов будет использовать уникальный ключ. Взгляните на следующий ниже пример, который это иллюстрирует:

```
CREATE TABLE sample (
 id INT NOT NULL,
 name VARCHAR(20),
 UNIQUE KEY(id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

Если разделение должно использовать уникальный ключ и столбец не имеет значения `NOT NULL`, то синтаксис разделения выдаст ошибку. Если в таблице определены первичные ключи, синтаксис разделения по ключу должен для создания разделов использовать первичный ключ. Взгляните на следующий ниже пример:

```
CREATE TABLE sample
(
 id INT NOT NULL PRIMARY KEY,
 name VARCHAR(20),
```

```
 email VARCHAR(100),
 UNIQUE KEY(email)
)
PARTITION BY KEY(id)
PARTITIONS 2;
```

В этом примере для разделения по ключу мы использовали первичный ключ.

Как и хеш-разделение, разделение по ключу тоже использует модульный алгоритм для создания разделения.

На таблице MySQL также можно задать линейное разделение по ключу. Вместо модульного алгоритма, используемого по умолчанию, в линейном разделении для создания упомянутых разделов будет использоваться мощность двух алгоритмов:

```
CREATE TABLE sample
(
 id INT NOT NULL PRIMARY KEY,
 name VARCHAR(20),
 email VARCHAR(100),
 UNIQUE KEY(email)
)
PARTITION BY LINEAR KEY(id)
PARTITIONS 2;
```

Если в разделении по ключу в качестве значения в столбце с определением раздела используется `NULL`, то `NULL` считается нулем.

Разбиение на подразделы

Как следует из названия, разбиение на подразделы (subpartitioning) делит разделенную таблицу MySQL на второй уровень разделения. Ниже приведен пример разбиения на подразделы:

```
CREATE TABLE employee (
 id INT NOT NULL PRIMARY KEY,
 joining_date DATE,
)
PARTITION BY RANGE(YEAR(joining_date))
SUBPARTITION BY HASH(TO_DAYS(joining_date))
SUBPARTITIONS 2 (
 PARTITION p0 VALUES LESS THAN (1990),
 PARTITION p1 VALUES LESS THAN (2000),
 PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

Таблицы, которые разделены с использованием диапазонов или списков, могут быть разделены далее. При разбиении на подразделы мы можем использовать метод хеш-разделения или разделения по ключу.

При использовании ключа в качестве метода разделения необходимо в качестве аргумента ключа передать значение уникального ключа или первичного ключа. Это вызвано тем, что при использовании ключа в качестве метода разбиения на подразделы MySQL не будет автоматически рассматривать как хеширующий ключ значение первичного или уникального столбца.

Кроме того, для каждого подраздела можно определить имя. При задании подразделов количество подразделов в каждом разделе должно быть одинаковым:

```
PARTITION BY RANGE(YEAR(joining_date))
SUBPARTITION BY HASH(TO_DAYS(joining_date))
SUBPARTITIONS 3 (
PARTITION partition1 VALUES LESS THAN (2005) (
 SUBPARTITION sub1,
 SUBPARTITION sub2,
 SUBPARTITION sub3,
),
PARTITION partition2 VALUES LESS THAN (2015) (
 SUBPARTITION sub4,
 SUBPARTITION sub5,
 SUBPARTITION sub6,
),
PARTITION partition2 VALUES LESS THAN MAXVALUE (
 SUBPARTITION sub7,
 SUBPARTITION sub8,
 SUBPARTITION sub9,
)
);
```

Таким образом, вы усвоили разные типы методов горизонтального разделения. При использовании любого из методов разделения он будет применяться как к таблице, так и к индексам. Это означает, что невозможно разделить только таблицу или только индексы. Кроме того, как отмечалось ранее, если таблица имеет первичный или уникальный ключ, то любое выражение столбцового разделения должно быть частью первичного или уникального ключа. Если первичный или уникальный ключ для таблицы не определен, то для разделения можно использовать любые другие столбцы.

Вертикальное разделение

Как отмечалось ранее, вертикальное разделение делит столбцы на несколько файловых групп.

Вертикальное разделение может быть достигнуто следующим образом:

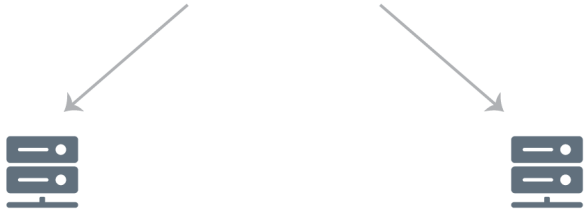
- разбиение строк на несколько файловых групп, которые могут храниться в нескольких местах;
- организация данных в несколько таблиц путем разбиения больших таблиц на более мелкие таблицы базы данных.

При строчном разбиении столбцы из одной таблицы выделяются по вертикали на несколько файловых групп. При наличии больших наборов данных, хранящихся в таблице, могут возникнуть трудности в управлении крупными файлами базы данных в одном расположении или на одном сервере. Поэтому мы можем распределить некоторые столбцы из базы данных в отдельную файловую группу, которой можно управлять на отдельном диске.

Рассмотрим следующий ниже снимок экрана с примером вертикального разделения пользовательской таблицы. Как отмечалось ранее, таблица базы данных `users` была разделена на два отдельных раздела, хранящихся в разных хранилищах.

В таблице может быть более двух разделов:

ID	User Name	First Name	Last Name	Email Address	Phone Number	Birth Date
1	MichellekDavis	Michellek	Davis	MichellekDavis@dayrep.com	9412233379	10/10/2086
2	dallasSGood	dallas	Good	DallasSGood@teleworm.us	8329843007	30/03/1972
3	TerranceDAviles	Terrance	Aviles	TerranceDaviles@armyspy.com	7045076659	20/04/1992
4	Undan1937	Susan	Roberts	SusanKRoberts@teleworm.us	4014126825	25/12/1996
5	Nempecovest1959	Lea	McGill	LeaJMcGill@jourrapide.com	2176567191	3/6/2000
6	Mandred	Regan	Coulter	ReganECoulter@jourrapide.com	6304832993	9/3/1998
7	Begivaing	Richard	Endres	RichardJEndres@rhyrep.com	3096848793	12/8/1988
8	Biturnight	Gregory	Melton	GregoryMMelton@dayrep.com	6304236158	6/9/1977
9	Stiong	Bernice	Stevenson	BerniceGStevenson@rhyrep.com	7736461308	3/1/2001



ID	User Name	First Name	Last Name	Email Address
1	MichellekDavis	Michellek	Davis	MichellekDavis@dayrep.com
2	dallasSGood	dallas	Good	DallasSGood@teleworm.us
3	TerranceDAviles	Terrance	Aviles	TerranceDaviles@armyspy.com
4	Undan1937	Susan	Roberts	SusanKRoberts@teleworm.us
5	Nempecovest1959	Lea	McGill	LeaJMcGill@jourrapide.com
6	Mandred	Regan	Coulter	ReganECoulter@jourrapide.com
7	Begivaing	Richard	Endres	RichardJEndres@rhyrep.com
8	Biturnight	Gregory	Melton	GregoryMMelton@dayrep.com
9	Stiong	Bernice	Stevenson	BerniceGStevenson@rhyrep.com

Phone Number	Birth Date
9412233379	10/10/2086
8329843007	30/03/1972
7045076659	20/04/1992
4014126825	25/12/1996
2176567191	3/6/2000
6304832993	9/3/1998
3096848793	12/8/1988
6304236158	6/9/1977
7736461308	3/1/2001

Вертикальное разделение одной таблицы на несколько файловых групп в MySQL не поддерживается. Однако мы можем добиться вертикального разделения путем нормализации данных. Если все столбцы в запросах не требуются часто, то всегда желательно разделить данные на несколько таблиц. Требования по соблюдению целостности данных могут быть достигнуты в MySQL за счет согласованности на нормализованных данных.

Разделение данных на многочисленные таблицы

Разделение данных – это процесс организации столбцов большой таблицы в несколько таблиц, которые в реляционной базе данных коррелируют с помощью ограничений «первичный ключ – внешний ключ». Крупные таблицы могут быть разделены таким образом, чтобы столбцы, которые не обновляются очень часто, могли быть перемещены в отдельную таблицу, и ссылочная целостность может быть реализована для этой таблицы.

Рассмотрим следующий ниже пример таблицы, которая имеет 10 столбцов.

Имя таблицы: users

user_id	first_name	last_name	email_address	profile_image	department	address_line1	address_line2	city	state

Теперь если мы посмотрим на эти 10 столбцов, то заметим, что есть определенные столбцы, которые могут быть NULL при вставке данных, поскольку они являются необязательными и могут не нуждаться в обновлении. Например, поле **profile_image** и адресные поля у пользователей подвержены частым изменениям. Поэтому из таблицы users мы можем создать три разные таблицы для хранения такой информации.

Имя таблицы: users

user_id	first_name	last_name	email_address	department

Имя таблицы: user_profile_image

profile_image_id	user_id	profile_image

Имя таблицы: user_address

address_id	user_id	address_line1	address_line2	city	state

Итак, теперь у нас есть три разные таблицы с пользовательской информацией, логически разделенной на столбцы таблицы. Эти три таблицы могут быть соединены путем определения внешних ключей на таблицах. Следующая ниже синтаксическая конструкция предназначена для создания трех таблиц пользователей с ограничениями по внешнему ключу для каждой таблицы:

```
CREATE TABLE IF NOT EXISTS users (
 user_id int(11) NOT NULL,
 first_name varchar(100) NOT NULL,
 last_name varchar(100) NOT NULL,
 email_address varchar(255) NOT NULL,
 department varchar(100) NOT NULL,
 PRIMARY KEY (user_id),
) ENGINE=InnoDB;
```

Как показано здесь, пользовательская таблица users создается с минимумом столбцов в таблице. Мы определили первичный ключ на столбце user_id, поэтому теперь мы можем связать другую пользовательскую информацию, используя ссылочную целостность.

Теперь давайте создадим таблицу user_profile_image:

```
CREATE TABLE IF NOT EXISTS user_profile_image (
 profile_image_id int(11) NOT NULL,
 user_id int(11) NOT NULL,
 profile_image blob NOT NULL,
 PRIMARY KEY (profile_image_id)
) ENGINE=InnoDB
```

Создадим таблицу user_address:

```
CREATE TABLE IF NOT EXISTS user_address (
 address_id int(11) NOT NULL,
 user_id int(11) NOT NULL,
 address_line1 VARCHAR(255),
```

```
 address_line2 VARCHAR(255),
 city VARCHAR(100),
 state VARCHAR(100),
 PRIMARY KEY (address_id)
) ENGINE=InnoDB;
```

Итак, теперь у нас есть три разные таблицы для хранения информации о пользователях. Каждая таблица содержит определенную область информации о пользователе, они полностью независимы друг от друга. Поэтому, когда вы захотите обновить информацию об адресе пользователя, MySQL не будет обновлять или проверять что-либо в схеме таблиц `users` или `user_profile_image`.

Если вы хотите удалить любую информацию о пользователе, то желательно удалить всю связанную информацию о пользователе. В MySQL это может быть достигнуто посредством ссылочной целостности. Рассмотрим следующую ниже синтаксическую конструкцию для добавления ссылочной целостности в таблицу `user_profile_image`:

```
ALTER TABLE user_profile_image
ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES `users`
(user_id) ON DELETE CASCADE ON UPDATE NO ACTION;
```

Данный запрос добавит связь между таблицами `users` и `user_profile_image`. MySQL 8 предоставляет некоторые готовые триггеры, которые могут использоваться при обновлении или удалении основных данных. Здесь оператор каскадного удаления `DELETE CASCADE` означает, что если запись в таблице пользователей удаляется, то она будет автоматически удалять связанные изображения из таблицы изображений, тем самым обеспечивая нам согласованность данных по всей базе данных. Аналогичные виды ограничений могут быть определены в таблице адресов.

Разбиение больших таблиц на несколько таблиц в целях разделения данных дает нам преимущества масштабирования данных относительно любых записей или документов. Например, мы переместили адресные данные в таблицу `user_address`. Теперь если есть необходимость разрешить пользователям иметь несколько адресов, то это может быть достигнуто без изменения конструкции базы данных.

При вертикальном разделении путем разделения данных на несколько таблиц требуется добавить некоторые соединения для извлечения всей информации по конкретной записи. Например, чтобы получить всю коррелирующую информацию о пользователе, необходимо выполнить следующий ниже запрос:

```
SELECT * FROM
  users
JOIN
  user_profile_image on users.user_id = user_profile_image.user_id
JOIN
  user_address on users.user_id = user_address.user_id
```

Таким образом, в сущности, чтобы получить всю информацию о пользователях, требуется объединить две таблицы. Перед вертикальным разделением данных на несколько таблиц необходимо нормализовать данные таким образом, чтобы все существующие данные были в наиболее нормальной форме.

Нормализация данных

Нормализация – это процесс организации данных таким образом, чтобы в таблице не было избыточных данных. В любой реляционной системе управления базами данных существует несколько уровней нормализации. В следующих далее разделах представлен краткий обзор различных уровней нормализации.

Первая нормальная форма

Когда в каждом столбце хранится уникальная информация, то говорят, что данные находятся в своей первой нормальной форме.

Например, в нашем предыдущем примере с пользователями, если какой-либо пользователь принадлежит нескольким отделам, то в отделе не должно быть значений, разделенных запятыми. Вместо этого для такой информации следует создать новую строку, чтобы таблица находилась в первой нормальной форме.

Когда в каждом столбце хранится уникальная информация, говорится, что данные находятся в своей первой нормальной форме.

Вторая нормальная форма

Для достижения второй нормальной формы таблица уже должна быть в своей первой нормальной форме, и все столбцы в таблице должны зависеть от уникального идентификатора данных.

Третья нормальная форма

Говорится, что данные находятся в третьей нормальной форме, если 2НФ уже достигнута и данные зависят только от ключа таблицы.

База данных считается нормализованной, если достигнута 3НФ.

Нормальная форма Бойса-Кодда

Таблица называется BCNF-таблицей (в нормальной форме Бойса-Кодда), если все реляционные таблицы могут быть идентифицированы с помощью одного первичного ключа из главной таблицы.

Четвертая нормальная форма

Если нет многозначных зависимостей от первичного ключа, то таблица находится в четвертой нормальной форме.

Пятая нормальная форма

Таблица находится в пятой нормальной форме, если данные не могут быть разделены на несколько таблиц без потери данных.

Подробное объяснение форм нормализации выходит за рамки этой книги.

При разбиении столбцов данных на несколько таблиц соединение двух или более таблиц может потребовать добавления в критерии соединения соответствующих столбцов, а также надлежащего индексирования столбцов. Более детально об определении индексов на столбцах можно узнать в главе 3 «*Индексирование данных для высокопроизводительных запросов*». Практические рекомендации в отношении соединения данных можно найти, обратившись к главе 7 «*Практические рекомендации по работе с СУБД MySQL 8*».

Подрезание разделов в MySQL

Подрезание (pruning) – это выборочное извлечение данных. Поскольку у нас есть многочисленные разделы больших данных, то во время извлечения они будут

проходить через каждый раздел, что отнимает много времени и влияет на производительность. Некоторые разделы также будут включены в поиск, в то время как запрошенные данные в этом разделе отсутствуют, что является избыточным процессом. Подрезание здесь помогает отыскать только те разделы, которые имеют соответствующие данные, позволяя избежать ненужного включения этих разделов во время извлечения.

Оптимизация, которая позволяет избегать сканирования разделов, где не может быть совпадающих значений, называется подрезанием разделов. При подрезании разделов оптимизатор анализирует операторы `FROM` и `WHERE` в запросах SQL для устранения ненужных разделов и сканирует те разделы базы данных, которые имеют отношение к запросу SQL. Давайте посмотрим на пример.

Предположим, что имеется таблица со следующей ниже структурой:

```
CREATE TABLE student (
 rollNo INT NOT NULL,
 name VARCHAR(50) NOT NULL,
 class CHAR NOT NULL,
 marks INT NOT NULL,
)
PARTITION BY RANGE(marks) (
 PARTITION p0 VALUES LESS THAN (35),
 PARTITION p1 VALUES LESS THAN (65),
 PARTITION p2 VALUES LESS THAN (85),
 PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Теперь, если мы хотим найти студентов, у которых все отметки меньше 35, запрос будет выглядеть следующим образом:

```
SELECT * FROM student WHERE marks < 35;
```

Понятно, что нам нужно искать только в разделе `p0`, и ни один из других разделов не должен быть включен для сканирования результата. Это вызвано тем, что учащиеся, получившие отметки менее 35, помещаются внутри раздела `p0`. Благодаря этому поиск в других разделах будет исключен, что занимает меньше времени и повышает производительность, по сравнению с поиском по всем разделам.

Следующий далее запрос будет производить поиск только в разделах `p1` и `p2`, а остальные будут проигнорированы:

```
SELECT * FROM student WHERE marks > 50 AND marks < 80;
```

Такой способ удаления разделов, которые не требуются во время сканирования, называется подрезанием. Подрезание разделов всегда будет быстрее обычного запроса, поскольку он будет сканировать необходимые разделы, а не искать все данные.

Подрезание может выполняться с помощью оператора `WHERE`. Можно использовать различные арифметические операторы, такие как `<`, `>`, `<=`, `>=`, `<>` и `=`. Если мы используем `=` (оператор равенства), то он определит, какой раздел содержит это значение, и только этот раздел будет проверен на соответствующие результаты. Мы также можем использовать предложения `BETWEEN` и `IN`. Это делается следующим образом:

```
SELECT * FROM student WHERE marks IN (30,50);
SELECT * FROM student WHERE marks BETWEEN 50 AND 80;
```

Оптимизатор вычислит выражение и определит раздел, в котором необходимо выполнить поиск. В первой инструкции оптимизатор узнает, что первое значение будет содержаться в разделе `p0`, а второе – в разделе `p1`. Поскольку `p2` и `p3` не содержат соответствующих значений, эти разделы при дальнейшем сканировании будут пропущены. Во втором запросе будет поиск только в разделах `p1` и `p2`.

Подрезание также может работать с таблицей, разделенной по `DATE`, `TIMESTAMP` или `DATETIME`. Давайте разберемся в этом на примере.

Предположим, у нас есть таблица, разделенная по дате `DATE` следующим образом:

```
CREATE TABLE access_log (
 log_id INT NOT NULL,
 type VARCHAR(100),
 access_url VARCHAR(100),
 access_date TIMESTAMP NOT NULL,
 response_time INT NOT NULL,
 access_by INT NOT NULL
)
PARTITION BY RANGE (UNIX_TIMESTAMP(access_date)) (
 PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2017-05-01 00:00:00')),
 PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2017-09-01 00:00:00')),
 PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2018-01-01 00:00:00')),
 PARTITION p3 VALUES LESS THAN (UNIX_TIMESTAMP('2018-05-01 00:00:00')),
 PARTITION p4 VALUES LESS THAN (UNIX_TIMESTAMP('2018-09-01 00:00:00')),
 PARTITION p5 VALUES LESS THAN (UNIX_TIMESTAMP('2019-01-01 00:00:00')),
);
```

Теперь давайте применим подрезание, исполнив следующий ниже запрос:

```
SELECT * FROM access_log WHERE access_date = UNIX_TIMESTAMP('2017-08-01 00:00:00');
```

В этом случае она будет сканировать только раздел `p1`, поскольку данные соответствующей даты включены только в этот раздел, а другие разделы игнорируются. Мы должны обеспечить формат даты, потому что если мы передадим недопустимый формат даты, то это вызовет `NULL`.

Подрезание разделов будет работать с запросами `SELECT`, `DELETE` и `UPDATE`, тогда как запрос `INSERT` подрезания разделов не поддерживает:

```
UPDATE access_log SET access_by = 10 WHERE access_date =
UNIX_TIMESTAMP('2017-08-01 00:00:00');

SELECT * FROM access_log WHERE UNIX_TIMESTAMP(access_date) =
UNIX_TIMESTAMP('2017-08-01 00:00:00');

DELETE FROM access_log WHERE UNIX_TIMESTAMP(access_date) =
UNIX_TIMESTAMP('2017-08-01 00:00:00');
```

Пока мы видели только примеры диапазонных разделов. Вас может заинтересовать вопрос, может ли подрезание применяться к другим видам разделения данных. Безусловно, оно может применяться ко всем другим разделам. Рассмотрим простой пример подрезания различных разделов.

Подрезание со списковым разделением

Подрезание может легко работать со списковым разделением для определения соответствующих разделов и сверяться только с ограниченными разделами. Предположим, у нас есть таблица с другой информацией о сотрудниках, которая выглядит следующим образом:

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT,
 store_id INT
)
PARTITION BY LIST(store_id) (
 PARTITION pNorth VALUES IN (3,5,6,9,17),
 PARTITION pEast VALUES IN (1,2,10,11,19,20),
 PARTITION pWest VALUES IN (4,12,13,14,18),
 PARTITION pCentral VALUES IN (7,8,15,16)
);
```

Теперь давайте выберем сотрудников по их магазинам с идентификаторами `(5, 6, 8)`:

```
SELECT * FROM employees WHERE store_id IN (5,6,8);
```

Оптимизатор может легко определить, в каком разделе содержатся значения `(5, 6, 8)`. Эти значения находятся в разделах `pNorth` и `pCentral`, а остальные разделы удалены из сканирования.

Подрезание с разделением по ключу

Раздел на основе ключа также может воспользоваться преимуществами подрезания для повышения производительности. Предположим, у нас есть таблица со следующей ниже информацией:

```
CREATE TABLE student (
 id INT NOT NULL PRIMARY KEY,
 name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 4;
```

Теперь давайте выберем информацию о студентах, идентификаторы которых содержатся между единицей и тремя:

```
SELECT * FROM student WHERE id > 1 AND id <3;
```

В этом запросе оператор `WHERE` будет преобразован в диапазон `WHERE id(1,2,3)` и будет применено подрезание. С разделами `HASH` и `KEY` подрезание может применяться только к целочисленным столбцам. Подрезание не будет работать, если мы разделили столбец `DATE`, использовав хеш-разделение `HASH` или разделение по ключу `KEY`.

Выполнение запросов на разделенных данных

Когда большие данные хранятся в MySQL, рекомендуется сделать разделы данных так, чтобы их можно было легко получать всякий раз, когда это необходимо. Это обусловлено тем, что выполнение запросов для неразделенных данных занимает слишком много времени. Разделение может быть полезно для исполнения запросов только на отобранных разделах вместо данных всей таблицы. Всегда быстрее выполнять запросы на нескольких фрагментах данных, чем на всем объеме. Существует два способа запросов, чье исполнение ускоряется с использованием разделения:

- **подрезание разделов**: как мы убедились в предыдущем разделе, подрезание может использоваться для извлечения данных только из определенных разделов. Эта работа автоматически определяет раздел, в котором должны быть запрошены данные;
- **предоставление выбранных разделов вместе с запросом**: вместе с исполняемым запросом можно явным образом задать имя раздела. Тем самым выбранный раздел может быть указан в запросах. Следующие операторы SQL поддерживают явное задание имени раздела:
  - SELECT;
  - INSERT;
  - UPDATE;
  - REPLACE;
  - DELETE;
  - LOAD DATA;
  - LOAD XML.

Ниже приведена синтаксическая конструкция, используемая в предыдущем запросе для явного указания имени раздела:

```
PARTITION (имена_разделов)
```

Здесь имена_разделов – это коллекция разделов, разделенных запятой. Этот параметр всегда следует за именем таблицы, к которой относится раздел в запросе. Мы можем указать имя раздела или подраздела указанной таблицы, и в случае если раздел не существует, будет выдаваться сообщение об ошибке: "partition 'partitionname' does not exist" («раздел» имя_раздела «не существует»).

Исполнение запроса с этим параметром проверит наличие только упомянутого раздела, в то время как остальные разделы будут проигнорированы. Рассмотрим пример использования этого параметра раздела.

Предположим, у нас есть сведения о результатах успеваемости студентов в соответствии со следующей структурой таблицы:

```
CREATE TABLE student (
 rollNo INT NOT NULL,
 name VARCHAR(50) NOT NULL,
 class CHAR NOT NULL,
 marks INT NOT NULL
)
PARTITION BY RANGE(marks) (
 PARTITION p0 VALUES LESS THAN (35),
 PARTITION p1 VALUES LESS THAN (65),
```

```
 PARTITION p2 VALUES LESS THAN (85),
 PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Теперь давайте вставим данные в таблицу `student`, как показано в следующем ниже примере:

```
INSERT INTO student VALUES
 (1, "Jaydip", "A", 80),
 (2,"Keyur","B", 30),
 (3,"Palak","A", 45),
 (4,"Malav","B", 84),
 (5,"Parth","C", 25),
 (6,"Kandarp","B",60),
 (7,"Chintan","C", 90);
```

Воспользуемся параметром `PARTITION`, чтобы получить записи только из раздела `p3`:

```
SELECT * FROM student PARTITION (p2);
```

```
mysql> SELECT * FROM student PARTITION (p2);
+--------+--------+-------+-------+
| rollNo | name   | class | marks |
+--------+--------+-------+-------+
|      1 | Jaydip | A     |    80 |
|      4 | Malav  | B     |    84 |
+--------+--------+-------+-------+
2 rows in set (0.00 sec)
```

Как вы можете видеть в предыдущем примере, он будет показывать только записи из раздела **p2**, в то время как остальные записи не отображаются. В одном запросе можно также указать несколько разделов. Это делается следующим образом:

```
SELECT * FROM student PARTITION (p1,p2);
```

```
mysql> SELECT * FROM student PARTITION (p1,p2);
+--------+---------+-------+-------+
| rollNo | name    | class | marks |
+--------+---------+-------+-------+
|      3 | Palak   | A     |    45 |
|      6 | Kandarp | B     |    60 |
|      1 | Jaydip  | A     |    80 |
|      4 | Malav   | B     |    84 |
+--------+---------+-------+-------+
4 rows in set (0.00 sec)
```

С помощью разделителя-запятой можно указать несколько разделов, и запрос будет извлекать записи из всех упомянутых разделов. Этот параметр может работать с любым типом раздела. У вас может возникнуть вопрос: как мы получаем имя раздела в разделении по ключу `KEY` или хеш-разделении `HASH`, поскольку мы не указываем имя раздела? В случае разделения `KEY` и `HASH` СУБД MySQL назначает имя каждого раздела автоматически, начиная с *p0, p1, p3, p4, ..., pN-1*, где *N* – количество разделов. Подразделам она присваивает имена *pXsp0, pXspl, pXsp2, ..., pXspN-1*, где *X* – это имя родительского раздела, а *N* – общее количество подразделов. Рассмотрим пример разделения по ключу `KEY`.

Предположим, у нас есть таблица, разделенная по ключу:

```
CREATE TABLE student1 (
 id INT NOT NULL PRIMARY KEY,
 name VARCHAR(20)
)
PARTITION BY KEY()
PARTITIONS 4;
```

Теперь давайте вставим данные в таблицу **student1**. В следующем ниже примере мы будем заполнять данные:

```
INSERT INTO student1 VALUES
 (1,"Jaydip"),
 (2,"Keyur"),
 (11,"Parth"),
 (12,"Palak"),
 (3,"Kandarp");
```

Здесь мы явно не указали имя раздела; следовательно, СУБД автоматически назначит `p0`, `p1`, `p2` и `p3` в качестве имен каждого раздела. Давайте проверим это, выбрав записи из первого раздела:

```
SELECT * FROM student1 PARTITION(p0);
```

```
mysql> SELECT * FROM student1 PARTITION(p0);
+----+---------+
| id | name    |
+----+---------+
|  1 | Jaydip  |
+----+---------+
1 row in set (0.00 sec)
```

Приведенный выше результат будет отображать все записи из раздела `p0`, который будет автоматически назначен в случае разделения по ключу.

До этого мы использовали разделение с запросом `SELECT`, но, как мы отмечали ранее, оно может работать и с другими запросами. Давайте воспользуемся этой возможностью с другими запросами.

Запрос DELETE с параметром PARTITION

Когда мы используем параметр `PARTITION` с инструкцией `DELETE`, он удаляет строки только из разделов или подразделов, упомянутых в запросе.

Предположим, что в группе A имеются следующие студенты:

```
mysql> SELECT * FROM student WHERE class = "A";
+--------+--------+-------+-------+
| rollNo | name   | class | marks |
+--------+--------+-------+-------+
|      3 | Palak  | A     |    45 |
|      1 | Jaydip | A     |    80 |
+--------+--------+-------+-------+
2 rows in set (0.00 sec)
```

Исполним следующий ниже запрос, чтобы удалить учащихся класса «A» из раздела `p2`:

```
DELETE FROM student PARTITION(p2) WHERE class = 'A';
```

```
mysql> SELECT * FROM student WHERE class = "A";
+--------+--------+-------+-------+
| rollNo | name   | class | marks |
+--------+--------+-------+-------+
|      3 | Palak  | A     |    45 |
|      1 | Jaydip | A     |    80 |
+--------+--------+-------+-------+
2 rows in set (0.00 sec)

mysql> DELETE FROM student PARTITION(p2) WHERE class = "A";
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM student WHERE class = "A";
+--------+-------+-------+-------+
| rollNo | name  | class | marks |
+--------+-------+-------+-------+
|      3 | Palak | A     |    45 |
+--------+-------+-------+-------+
1 row in set (0.00 sec)
```

Как показано в предыдущем результате, запрос удалил соответствующие записи из раздела **p2**, а остальные разделы этим запросом затронуты не были. Другие данные о группе «A» по-прежнему существуют, но они принадлежат другим разделам.

Запрос UPDATE с параметром PARTITION

Оператор `UPDATE` ведет себя так же, как и другие операторы с параметром `PARTITION`. Он будет обновлять записи только выбранного раздела. Давайте обновим столбец учебной группы 'A' в таблице студентов только для раздела `p1` в случае, если их отметки выше 50 баллов:

```
UPDATE student PARTITION(P1) SET class = 'A' WHERE marks > 50;
```

Взгляните на следующий ниже результат, который показывают записи перед и после исполнения запроса на обновление:

```
mysql> SELECT * FROM student PARTITION(p1);
+--------+---------+-------+-------+
| rollNo | name    | class | marks |
+--------+---------+-------+-------+
|      3 | Palak   | A     |    45 |
|      6 | Kandarp | C     |    60 |
+--------+---------+-------+-------+
2 rows in set (0.00 sec)

mysql> UPDATE student PARTITION(P1) SET class = 'A' WHERE marks > 50;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM student PARTITION(p1);
+--------+---------+-------+-------+
| rollNo | name    | class | marks |
+--------+---------+-------+-------+
|      3 | Palak   | A     |    45 |
|      6 | Kandarp | A     |    60 |
+--------+---------+-------+-------+
2 rows in set (0.01 sec)
```

Запрос INSERT с параметром PARTITION

Параметр `PARTITION` с оператором `INSERT` при вставке записей в определенный раздел ведет себя немного по-другому. Он выполняет проверку, подходит ли соответствующая запись для раздела. Если запись не подходит для раздела, то он вызовет

сообщение об ошибке «несоответствие заданному набору разделов» (*Not matching the given partition set*). Вставим запись в раздел `p2` таблицы `student`:

```
INSERT INTO student PARTITION(P2) VALUES(7,'Jaydip','A',70);
```

Как показано в следующем ниже результате, в первый раз это вызвало ошибку, потому что мы попытались сохранить отметки 30 в разделе **p2**, в то время как этот раздел содержит отметки только между 65 и 85. После этого, когда мы попытались сохранить отметки 70, они были успешно вставлены. Следовательно, прежде чем вставлять определенную запись, этот оператор всегда проверяет, может раздел содержать запись или нет.

```
mysql> SELECT * FROM student PARTITION(p2);
+--------+-------+-------+-------+
| rollNo | name  | class | marks |
+--------+-------+-------+-------+
|      4 | Malav | B     |    84 |
+--------+-------+-------+-------+
1 row in set (0.00 sec)

mysql> INSERT INTO student PARTITION(P2) VALUES(7,'Jaydip','A',30);
ERROR 1748 (HY000): Found a row not matching the given partition set
mysql> INSERT INTO student PARTITION(P2) VALUES(7,'Jaydip','A',70);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM student PARTITION(p2);
+--------+--------+-------+-------+
| rollNo | name   | class | marks |
+--------+--------+-------+-------+
|      4 | Malav  | B     |    84 |
|      7 | Jaydip | A     |    70 |
+--------+--------+-------+-------+
2 rows in set (0.00 sec)
```

В сценарии использования нескольких разделов с оператором `INSERT` запись будет вставлена в соответствующий раздел, который соответствует критериям раздела. Кроме того, если ни один из разделов не соответствует ни одной записи, то вся вставка завершится аварийно и вызовет ошибку.

Резюме

В этой главе вы познакомились с горизонтальным и вертикальным разделением. В ней были рассмотрены различные типы горизонтального разделения. Затем мы разобрались с тем, как достичь наилучших из имеющихся в MySQL преимуществ вертикального разделения. Вы также изучили методы подрезания и запросы на извлечение значений из нескольких разделов. Мы обсудили вопросы удаления и изменения разделов в MySQL. В следующей главе мы рассмотрим несколько интересных тем. Вы познакомитесь с тем, каким образом можно улучшить производительность MySQL, используя репликации, и с различными методами репликации, имеющимися в MySQL 8.

Глава 6

Репликация для построения высокодоступных решений

В предыдущей главе вы узнали, что такое разделение, познакомились с типами разделения и выполнением запросов на разделенных данных, которые помогают повысить производительность.

В этой главе мы собираемся обсудить следующие ниже темы в деталях:

- высокая доступность;
- репликация;
- групповая репликация.

Высокая доступность

Сегодня данные являются важнейшей частью любого веб-, мобильного, социального, корпоративного и облачного приложения. Обеспечение доступности данных всегда является главным приоритетом для любого приложения. Небольшое время простоя может привести к значительной потере дохода и репутации организации.

Постоянная доступность данных стала очень критичной в настоящее время. И поскольку размер данных со временем увеличивается, может оказаться сложным поддерживать огромный объем данных, доступных в любой момент времени. Потеря связи с приложением на долю секунды также может оказать огромное влияние на организацию и может поставить на карту ее репутацию.

Чтобы избежать влияния несогласованности в доступе к приложениям из-за базы данных, требуется потратить время на создание высокодоступной архитектуры. Для создания высокодоступной инфраструктуры приложений необходимо учитывать многие вещи, такие как возможности технологий, используемых для создания приложений, потребность бизнеса в создании такой инфраструктуры и т. д. Это, безусловно, оказывает сильное влияние, следовательно, мы должны эти вещи учитывать, прежде чем делать какие-либо выводы о создании высокодоступной архитектуры.

Для достижения высокой доступности и подготовки полностью масштабируемой системы MySQL 8 имеет все возможности, предоставляя приложению соответствующую серверную часть. Способность системы поддерживать постоянное подключение в случае аварийного завершения работы части инфраструктуры и способность системы восстанавливаться после такого аварийного завершения

работы называется высокой доступностью. Аварийное завершение работы системы может быть вызвано либо деятельностью по обслуживанию на одной из частей системы, такой как обновление аппаратного или программного обеспечения, либо вследствие аварийного завершения работы установленного программного обеспечения.

Если рассматривать масштабируемость с точки зрения MySQL 8, то MySQL 8 поддерживает масштабируемость посредством репликации базы данных по многочисленным серверам MySQL с целью разделения нагрузки, вызываемой запросами от приложения.

Требование по достижению высокой доступности и масштабируемости может разниться от системы к системе. Для того чтобы достигнуть этих свойств, каждая система требует разной конфигурации. MySQL 8 также поддерживает различные подходы, такие как репликацию данных через многочисленные серверы MySQL или подготовку многочисленных центров обработки данных на основе географических местоположений и обслуживания запросов клиентов из центров обработки данных, ближайших к местоположению клиента. Подобные решения могут быть использованы для достижения максимального периода безотказной работы MySQL.

Атрибуты, которые имеют значение при выборе верного решения для высокой доступности, зависят от того, до какой степени система может называться высокодоступной, поскольку такое требование варьируется от системы к системе. Для небольших приложений, где пользовательская нагрузка не ожидается очень высокой, настройка среды репликации или кластера может привести к очень высокой стоимости. В таких случаях может быть достаточно просто обеспечить правильную конфигурацию MySQL и тем самым уменьшить нагрузку на приложение. Мы узнаем подробности о настройке правильной конфигурации MySQL в главе 7 «*Практические рекомендации по работе с MySQL 8*».

Следующие далее разделы знакомят с первичными решениями, поддерживаемыми MySQL 8 для достижения высокой доступности, и их краткими подробностями.

Репликация в MySQL

Репликация MySQL позволяет тиражировать данные из одного сервера на многочисленные серверы MySQL. Репликация MySQL обеспечивает структуру типа «ведущий-ведомый» (master-slave), в которой один из серверов группы действует как ведущий, где операции записи выполняются из приложения, и затем ведущий сервер копирует данные на многочисленные ведомые серверы. Репликация является хорошо зарекомендовавшим себя решением для обеспечения высокой доступности и используется такими социальными гигантами, как Facebook, Twitter и т. д.

Кластер MySQL

Это еще одно популярное решение MySQL для обеспечения высокой доступности. Кластеризация позволяет реплицировать данные на несколько серверов MySQL с автоматическим совместным использованием. Оно спроектировано для более высокой доступности и пропускной способности.

Облачная служба Oracle MySQL

Облачная служба Oracle MySQL предоставляет службу баз данных MySQL, которая позволяет организациям быстро, безопасно и экономично применять современные приложения для работы с самой популярной в мире системой управления базами данных с открытым исходным кодом.

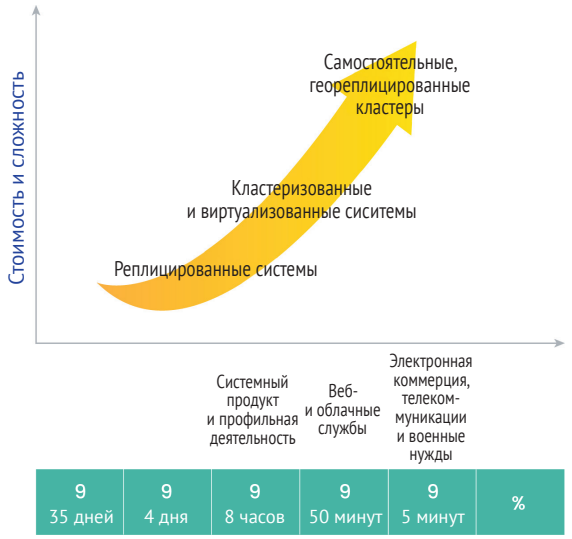
MySQL с кластером Solaris

Кластер Sun Solaris HA для службы данных MySQL предоставляет механизм упорядоченного запуска и завершения работы, мониторинга сбоя и автоматического аварийного переключения службы MySQL. Следующие ниже компоненты MySQL имеют защиту Sun Cluster HA для служб данных MySQL.

Есть еще несколько вариантов с применением сторонних решений. Каждая архитектура, используемая для обеспечения высокой доступности служб баз данных, различается по уровням времени безотказной работы. Эти архитектуры можно сгруппировать по трем основным категориям. На следующей ниже схеме показана архитектура каждой категории:

- репликация данных;
- кластеризованные и визуализованные системы;
- географически реплицированные кластеры.

Ниже приведен рисунок, иллюстрирующий три разных метода и уровень безотказной работы, который может быть достигнут путем его реализации.



Согласно схеме, эти три метода следующие:

- репликация данных на многочисленные серверы MySQL через групповую репликацию MySQL 8;
- настройка MySQL на виртуальных серверах и построение кластеризованной виртуальной среды;
- географически ориентированная репликация данных, где каждый сервер в другом географическом местоположении обслуживает отдельную группу клиентов и не имеет общих данных между ними.

Сложность системы так же, как и стоимость, может сыграть большую роль в выборе любого из этих трех методов. Поэтому, прежде чем выбрать любой из трех методов, упомянутых на предыдущем рисунке, необходимо задать себе определенный вопрос:

- Какие данные для вас важны, и насколько они важны?
- Какое влияние окажет на приложение потеря некоторых данных?
- Что будет влиять на приложение, если все данные будут потеряны? Может ли восстановление какой-либо исторической резервной копии послужить достижению вашей цели или нет?
- Каково будет влияние, если вы не сможете получить доступа к базе данных в течение минуты, часа, дня или недели?

Опираясь на самый лучший ответ на вопрос, вы сможете выбрать правильный вариант для вашего приложения, с оптимальной стоимостью и высокодоступным решением. Это обсуждение дает нам достаточный обзор высокой доступности MySQL 8. Теперь давайте подробно обсудим репликацию и каким образом можно настроить различные типы репликации.

Репликация с помощью MySQL

Репликация MySQL работает на основе архитектуры ведущий-ведомый. Один из группы серверов MySQL действует как ведущий, а другие серверы работают как ведомые серверы. Все операции записи выполняются на ведущем сервере, в то время как операции чтения выполняются на любом из ведомых серверов. Связь между ведущим и ведомыми серверами поддерживается на основе конфигурации. В следующих темах мы узнаем различные конфигурации для синхронизации ведущей и ведомых баз данных.

Преимущества репликации в MySQL 8

Давайте кратко рассмотрим несколько преимуществ репликации MySQL 8.

Масштабируемые приложения

Как мы уже говорили ранее, репликация разделяет нагрузку между ведущей и ведомыми базами данных. Поэтому теперь объем работы базы данных уменьшился. Ведущая база данных будет выполнять только операции записи и не будет беспокоиться об операциях чтения, в то время как ведомые базы данных будут заботиться только об операции чтения, не беспокоясь об операциях записи. Этот механизм фактически приводит к более быстрой работе MySQL. Кроме того, новые серверы MySQL легко добавляются к репликационной группе, тем самым масштабируя архитектуру вширь в случае большой нагрузки на сервер.

Безопасная архитектура

Одно из преимуществ, которое мы можем получить благодаря такой репликации, состоит в том, что один из ведомых серверов может использоваться для резервного копирования, избавляя от выполнения резервного копирования на ведущем сервере MySQL. Это делает процесс резервного копирования независимым от приложения, использующего MySQL 8. При этом не будет никакого влияния на операцию записи ведущего сервера. Проще говоря, это исключает вероятность повреждения данных во время резервного копирования.

Анализ крупных данных

Поскольку операции чтения выполняются на ведомых серверах, мы можем назначить одному из ведомых серверов выполнение сложных запросов чтения, что позволяет генерировать отчеты о выполненном анализе данных на MySQL 8, так как выполнение сложных запросов чтения не повлияет на общую производительность приложения.

Географически совместное использование данных

Групповая репликация позволяет копировать данные ведущего сервера на ведомый сервер, размещенный в удаленном местоположении и выполняющий операции чтения для отдельной группы клиентов без влияния на операции ведущего сервера.

Методы репликации в MySQL 8

Давайте рассмотрим различные методы репликации, имеющиеся в MySQL 8.

Репликация с использованием двоичных журналов

С опорой на репликацию событий из сгенерированных ведущим сервером двоичных журналов они будут синхронизированы с ведущим и ведомым серверами. Этот метод поддерживается в MySQL 5.1, однако в MySQL 8 у него есть много новых и захватывающих функциональных возможностей, которые будут раскрыты в главе 7 «*Практические рекомендации по работе с MySQL 8*».

Репликация с использованием глобальных идентификаторов транзакций

Глобальные идентификаторы транзакций (GTID) вместо репликации на основе двоичных журналов используют транзакционно-ориентированную репликацию данных. До тех пор, пока транзакции, которые находились в работе и были зафиксированы на ведущих серверах, не присутствуют на всех ведомых серверах, глобальный идентификатор GTID не будет рассматривать репликацию как находящуюся в согласованном состоянии.

В MySQL 8 репликация может выполняться либо в асинхронном, либо в полусинхронном режиме. В асинхронном режиме операции записи выполняются на ведущем сервере немедленно, в то время как репликация на ведомых серверах выполняется периодически согласно конфигурации.

В полусинхронном режиме репликации, если полусинхронная конфигурация активирована на ведущем сервере и, по крайней мере, одном ведомом сервере, транзакция на ведущем узле ожидает до достижения тайм-аута получения транзакции до тех пор, пока полусинхронно активированный узел не подтвердит, что необходимые данные или обновление были получены. И на тайм-ауте ведущий узел снова отыскивает полусинхронный сервер и выполняет репликацию.

Конфигурация репликации

Теперь давайте подробно разберемся в конфигурации различных методов репликации, имеющихся в MySQL 8:

1) репликация с двоичными журналами;
2) репликация с глобальными идентификаторами GTID;
3) многоисточниковая репликация в MySQL 8.

Кроме того, мы выясним различные конфигурации, доступные для настройки репликации на серверах.

Репликация с двоичными журналами

Файл двоичного журнала регистрации событий содержит записи всех операций MySQL, выполненных в MySQL. Каждая операция хранится в разном формате. Поэтому для репликации на основе двоичного журнала ведущий сервер выполняет операции вставки и обновления, которые отражаются в двоичном журнале. Далее ведомые узлы конфигурируются для чтения этих двоичных файлов, и те же самые события выполняются в двоичном файле ведомых серверов для репликации данных на ведомые серверы.

Копия всего двоичного журнала передается ведущим сервером всем ведомым серверам. По умолчанию все события из ведущего сервера выполняются на ведомом сервере. Мы можем сконфигурировать ведомые серверы для обработки событий, которые принадлежат только какой-либо конкретной базе данных или таблице. Мы также можем сконфигурировать на ведомом сервере обработку конкретных событий из ведущего сервера, которые будут выполнены на ведомом сервере.

При исполнении событий из двоичного журнала, предоставленного ведущим сервером, на ведомом сервере на последнем лежит ответственность помнить позицию в файле, где он выполнил последнее выполнение. Ведомый сервер может быть сконфигурирован для периодического чтения двоичного журнала. Поэтому, если имеется более одного ведомого сервера, каждый из них будет иметь собственный ряд операций, которые он должен выполнить на следующем запланированном выполнении. Все серверы, используемые в репликации, или ведущий сервер, или ведомый сервер, содержат свой собственный уникальный идентификатор, который называется идентификатором сервера `server-id`. При чтении двоичного журнала из ведущего узла ведомые серверы содержат информацию, связанную с ведущим сервером, такую как имя хоста, имя журнального файла и позиция внутри файла.

Давайте обсудим, как настроить репликацию. Мы исходим из того, что в целях репликации у вас уже есть два сервера MySQL. В этом примере будут использоваться следующие ниже IP-адреса сервера MySQL:

- 12.34.56.789: ведущая база данных;
- 12.23.34.456: ведомая база данных.

Конфигурация репликации на ведущем сервере

Откройте конфигурационный файл MySQL на ведущем сервере:

```
sudo vim /etc/mysql/my.cnf
```

Внутри этого файла мы должны внести следующие изменения. Первый шаг – найти раздел, который выглядит следующим образом. Здесь мы связываем сервер с localhost:

```
bind-address= 127.0.0.1
```

Заменим стандартный IP-адрес IP-адресом ведущего сервера:

```
bind-address= 12.34.56.789
```

Эта замена позволяет ведомому серверу MySQL обращаться к базам данных с помощью IP-адреса.

Теперь мы должны сконфигурировать уникальный идентификатор сервера для ведущего сервера и указать путь к двоичному журналу на ведущем сервере. Убедитесь, что следующая ниже строка раскомментирована:

```
server-id= 1
log-bin= /var/log/mysql/mysql-bin.log
```

Здесь `mysql-bin.log` – имя файла, где хранятся все регистрационные записи.

Наконец, нам нужно сконфигурировать базу данных, которая будет реплицироваться на ведомом сервере. Можно включить более одной базы данных, повторяя эту строку для всех баз данных, в которых вы нуждаетесь:

```
binlog_do_db= shabbirdb
```

`shabbirdb` – это имя базы данных. После внесения изменений перезапустите сервер:

```
sudo service mysql restart
```

Далее следующие ниже шаги предоставляют привилегии ведомому серверу. Открываем командную строку MySQL:

```
mysql -u root -p
```

Создаем пользователя MySQL, который будет применяться ведомым сервером, и применяем инструкцию SQL `GRANT`, чтобы предоставить доступ пользователю, который будет пользователем из ведомого сервера:

```
CREATE USER 'shabbirslave'@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO 'dbslave'@'%' IDENTIFIED BY 'password';
```

Очищаем кеш привилегий с помощью следующей ниже команды:

```
FLUSH PRIVILEGES;
```

Далее переключаемся на базу данных, которую требуется реплицировать:

```
USE shabbirdb;
```

Возьмите свою резервную копию базы данных с помощью команды `mysqldump`, которую мы будем использовать для создания ведомого сервера. Перед резервным копированием заблокируйте базу данных и проверьте текущую позицию базы данных, которую вы будете использовать позже при создании ведомого сервера:

```
FLUSH TABLES WITH READ LOCK;
```

Ниже приведена команда для проверки состояния ведущего сервера:

```
SHOW MASTER STATUS;
```

Результат этой команды покажет имя двоичного файла и позицию, с которой он начнет чтение.

```
mysql> SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      102 | shabbirdb    |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Как показано в полученном результате, вывод команды покажет местоположение внутри журнального файла, откуда ведомый сервер начнет репликацию. `mysql-bin.000001` – это имя журнального файла, который читается на ведомом сервере. Мы будем использовать это имя журнального файла и позицию в нем в качестве ссылки в дальнейших примерах.

Для дальнейшей конфигурации мы будем использовать базу данных непосредственно в самом режиме блокировки. Любая дальнейшая команда за один сеанс разблокирует существующие блокировки. Поэтому необходимо, чтобы предстоящая команда была применена в отдельном сеансе MySQL.

Теперь нам нужно экспортировать базу данных с помощью команды MySQL `mysqldump` с активированным режимом блокировки.

```
mysqldump -u root -p shabbirdb > shabbirdb.sql
```

После того как база данных экспортирована, мы можем разблокировать базу данных:

```
UNLOCK TABLES;
QUIT;
```

Таким образом, выполнив все предыдущие шаги, мы закончили с конфигурацией ведущего сервера для репликации. Теперь следующим шагом будет конфигурирование ведомого сервера.

Конфигурация репликации на ведомом сервере

Теперь, когда ведущий сервер сконфигурирован, давайте подключимся к ведомому серверу MySQL и создадим базу данных, которую мы будем использовать для репликации в качестве ведомой.

```
CREATE DATABASE shabbirdb;
quit;
```

База данных, которую мы экспортировали в режиме блокировки, должна быть импортирована сюда, на ведомые серверы.

```
mysql -u root -p shabbirdb < /path/to/shabbirdb.sql
```

Теперь нам нужно настроить конфигурацию ведомого сервера. Открывем файл `my.cnf` ведомого сервера:

```
sudo vim /etc/mysql/my.cnf
```

Теперь нам нужно предоставить уникальный `server-id` ведомому серверу:

```
server-id = 2
```

Затем нам нужно сконфигурировать `relay-log`, `log_bin` и `binlog_do_db` в файле `my.cnf`:

```
relay-log = /var/log/mysql/mysql-relay-bin.log
log_bin = /var/log/mysql/mysql-bin.log
binlog_do_db = newdatabase
```

Перезапускаем MySQL:

```
sudo service mysql restart
```

Следующий шаг – активировать репликацию внутри командной оболочки MySQL.

Открываем командную оболочку MySQL и набираем следующие ниже сведения, заменив значения в соответствии с информацией, записанной из ведущего сервера:

```
CHANGE MASTER TO MASTER_HOST='12.34.56.789',MASTER_USER='shabbirslave',
MASTER_PASSWORD='password', MASTER_LOG_FILE='mysql-bin.000001',
MASTER_LOG_POS= 103;
```

Эта команда выполняет несколько действий одновременно:

- она обозначает текущий сервер как ведомый нашим ведущим сервером;
- она предоставляет серверу правильные учетные данные для входа;
- она дает знать ведомому серверу, где начинать реплицировать из ведущего журнального файла, и журнальная позиция (103) поступает из чисел, которые мы записали ранее.

Теперь активируем ведомый сервер:

```
START SLAVE;
```

Репликация с глобальными идентификаторами GTID

Используя этот метод, каждая транзакция может идентифицироваться и отслеживаться по мере ее фиксации на ведущем сервере и применения на ведомом сервере. Этот метод не ссылается на журнальные файлы или позиции внутри этих файлов во время запуска нового ведомого сервера или аварийного перехода к новому ведущему серверу. Репликация на основе GTID полностью основана на транзакциях; она позволяет легко определить, согласованы ли главный и ведомый серверы. Поскольку все транзакции, зафиксированные на ведущем сервере, также фиксируются на ведомом сервере, согласованность между обоими гарантируется.

Использование глобального транзакционного идентификатора дает два основных преимущества:

- **легко заменить ведущий сервер для соединения с ведомым сервером во время аварийного переключения**: GTID уникален для всех серверов в репликационной группе. Ведомый сервер помнит глобальный идентификатор GTID последнего события из старого ведущего сервера. Это позволяет легко узнавать, где возобновлять репликацию на новом ведущем сервере, поскольку глобальные транзакционные идентификаторы известны во всей репликационной иерархии;
- **состояние ведомого сервера записывается аварийно-безопасным способом**: ведомый сервер отслеживает свою текущую позицию в системной таблице `mysql.gtid_slave_pos`. Если эта таблица использует транзакционную подсистему хранения данных (такую как InnoDB, которая принята по умолчанию), то обновления в состоянии делаются в той же транзакции, что и обновления данных. Это делает состояние аварийно-безопасным; если ведомый сервер аварийно завершает работу, аварийное восстановление при перезапуске удостоверится, что записанная позиция репликации совпадает с изменениями, которые были реплицированы. Это не относится к классической репликации, где состояние записывается в файл `relay-log.info`, который обновляется независимо от фактических изменений в данных и может легко выйти из синхронизации, если ведомый сервер аварийно завершает работу.

Прежде чем мы перейдем к конфигурационной части, давайте получше разберемся в GTID.

Глобальные идентификаторы транзакций

Глобальный идентификатор транзакции GTID – это уникальный ключ, созданный и связанный с каждой транзакцией (операцией вставки и обновления), зафиксированной на ведущем сервере. Ключ не только уникален для ведущего сервера, но и уникален для всех серверов в репликации. Существует взаимно-однозначное соответствие между всеми транзакциями и всеми глобальными идентификаторами GTID.

GTID представлен в виде пары источниковых идентификаторов и транзакционных идентификаторов, разделенных символом двоеточия (:):

```
GTID = источниковый_идентификатор:транзакционный_идентификатор
```

`источниковый_идентификатор` – это `server_uuid` ведущего сервера. `транзакционный_идентификатор` – это порядковый номер, определяемый порядком, в котором транзакция была зафиксирована на сервере. Например, 23-я транзакция, которая первоначально будет зафиксирована на сервере с UUID `17ede83a-8b22-11e7-8d2e-d0bf9c10f6c8`, имеет такой глобальный идентификатор GTID:

```
17ede83a-8b22-11e7-8d2e-d0bf9c10f6c8:23
```

Этот формат используется с инструкцией `START SLAVE` в качестве аргумента для конфигурации ведомой базы данных. Данный формат используется для представления глобальных идентификаторов GTID в результатах таких инструкций, как `SHOW SLAVE STATUS`, а также в двоичном журнале.

Важные переменные GTID следующие:

- **gtid-mode**: ON|OFF;
- **enforce-gtid-consistency**: предотвращает исполнение транзакционно небезопасных инструкций;
- **gtid-purged**: транзакции были удалены из двоичных журналов;
- **gtid-executed**: транзакции уже исполнены на сервере;
- **gtid-next**: глобальный идентификатор GTID для следующей транзакции.

Генерация глобального идентификатора GTID состоит из следующих шагов.

1. Когда транзакция исполняется и фиксируется на ведущем сервере, эта транзакция назначается глобальному идентификатору GTID. Этот GTID записывается в двоичный журнал ведущего сервера.
2. После того как данные двоичного журнала переданы на ведомый сервер и сохраняются в передаточном журнале ведомого сервера, ведомый сервер читает глобальный идентификатор GTID и назначает этот GTID системной переменной `gtid_next`. Это сообщает ведомому серверу, что следующая транзакция должна быть зарегистрирована в журнале, используя данный GTID.
3. Ведомый сервер проверяет, что этот глобальный идентификатор GTID еще не использовался для регистрации транзакции в ее собственном двоичном журнале. Если этот GTID не был использован, ведомый сервер тогда записывает GTID, применяет транзакцию и записывает транзакцию в свой двоичный журнал. В первую очередь читая и проверяя GTID транзакции перед выполнением, ведомый сервер гарантирует не только то, что никакая предыдущая транзакция с этим GTID не применялась на ведомом сервере, но

также то, что никакая иная сессия не прочитала этот GTID и при этом не выполнила связанную транзакцию. Другими словами, нескольким клиентам не разрешается применять одну и ту же транзакцию одновременно.

4. Поскольку переменная `gtid_next` не пуста, ведомый сервер не пытается генерировать GTID для этой транзакции, а вместо этого записывает GTID, сохраненный в этой переменной, то есть GTID, полученный от ведущего сервера, непосредственно предшествующий транзакции в своем двоичном журнале.

Таблица gtid_executed

Эта таблица MySQL принята по умолчанию. Она содержит интервал нескольких глобальных идентификаторов GTID. Эта таблица содержит следующие столбцы:

- `source_uuid CHAR(36) NOT NULL`: идентификатор `uuid` ведущего сервера, где транзакция исходно исполнилась;
- `interval_start BIGINT NOT NULL`: первое число интервала;
- `interval_end BIGINT NOT NULL`: последнее число интервала.

Если режим `gtid_mode` активирован, то она хранит глобальные идентификаторы GTID в таблице `gtid_executed`, а если `binlog` деактивирован, то она хранит глобальные идентификаторы GTID, принадлежащие транзакции.

Конфигурации GTID на стороне ведущего сервера

Откройте конфигурационный файл MySQL на ведущем сервере:

```
sudo vim /etc/mysql/my.cnf
```

Внутри этого файла мы должны внести следующие изменения:

```
[mysqld]
server-id = 1
log-bin = mysql-bin
binlog_format = ROW
gtid_mode = on
enforce_gtid_consistency
log_slave_updates
```

Перезапустите MySQL, чтобы применить изменения конфигурации:

```
sudo service mysql restart
```

Создайте пользователя MySQL, который будет применяться ведомым сервером, и используйте иснтрукцию SQL `GRANT`, чтобы обеспечить доступ для пользователя, который будет использоваться ведомым сервером:

```
CREATE USER 'shabbirslave'@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO 'shabbirslave'@'%' IDENTIFIED BY 'password';
```

Возьмите резервную копию базы данных с помощью команды `mysqldump`, которая используется для создания ведомого сервера:

```
mysqldump -u root -p shabbirdb > shabbirdb.sql
```

Конфигурации GTID на стороне ведомого сервера

Итак, поскольку мы сконфигурировали ведущий сервер, теперь давайте сконфигурируем ведомый сервер. Нам нужно будет создать базу данных на ведомом сервере, которую мы экспортировали из ведущего сервера.

Импортируйте базу данных, ранее экспортированную из ведущей базы данных:

```
mysql -u root -p shabbirdb < /path/to/shabbirdb.sql
```

Добавьте следующие ниже переменные в `my.cnf`:

```
[mysqld]
server_id        = 2
log_bin          = mysql-bin
binlog_format    = ROW
skip_slave_start
gtid_mode        = on
enforce_gtid_consistency
log_slave_updates
```

Перезапустите MySQL, чтобы применить изменения конфигурации:

```
sudo service mysql restart
```

Исполните команду `CHANGE MASTER TO`:

```
CHANGE MASTER TO
MASTER_HOST='12.34.56.789',
MASTER_PORT=3306,
MASTER_USER='shabbirslave',
MASTER_PASSWORD='password',
MASTER_AUTO_POSITION=1;
```

Начните репликацию:

```
START SLAVE;
```

Важно указать позицию, с которой ведомый сервер начнет процесс синхронизации. Это вызвано тем, что `mysqldump` имеет приведенную выше информацию о GTID. Эта переменная копируется в ведомый сервер во время импорта дампа и используется для синхронизации базы данных.

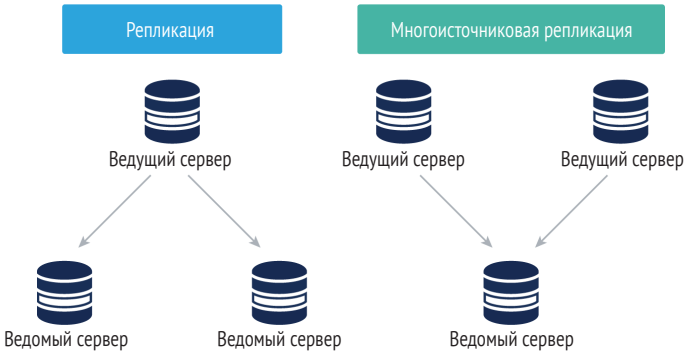
```
-- -- GTID state at the beginning of the backup --
SET @@GLOBAL.GTID_PURGED='b9b4712a-df64-11e3-b391-60672090eb04:1-7';
```

Многоисточниковая репликация MySQL

Многоисточниковый MySQL сервер является обратным традиционной репликации из ведущего сервера на ведомый. В одном механизме с одним ведущим сервером и множеством ведомых многочисленные ведомые серверы соединены с одиночным ведущим сервером для операции чтения. В многоисточникой репликации MySQL существует несколько ведущих баз данных, где выполняются операции записи. И один одиночный ведомый сервер соединяется с многочисленными ведущими базами данных и реплицирует данные из всех ведущих баз данных параллельно.

Как мы узнали, многоисточниковая репликация MySQL может подключаться к более чем одному ведущему серверу и выполнять операцию репликации параллельно и имеет много преимуществ. В то время как репликация «ведущий-ведомый» рассматривается на предмет высокой доступности и масштабируемости, многоисточниковая репликация может рассматриваться на предмет операции резервного копирования или сбора информации из более чем одного сервера для

создания сводного отчета. Любой конфликт в транзакциях, который возникает во время репликации из многоисточниковых серверов, не может быть обнаружен или разрешен MySQL 8. Любые такие конфликты в транзакциях должны быть приняты во внимание самим приложением. Для репликации ведомый сервер создает один отдельный канал для каждого сервера-источника, к которому он подключен для выполнения операции репликации.



Конфигурация многоисточниковой репликации

Ведущие серверы могут быть сконфигурированы с помощью глобальных идентификаторов транзакций. Мы рассмотрели конфигурацию ведущего узла в этой главе, когда обсуждали *репликацию с использованием глобальных идентификаторов транзакций*. Ведомые серверы в многоисточниковой репликации не могут работать на файловых хранилищах. MySQL поддерживает конфигурацию либо с файловыми, либо с табличными хранилищами. Поэтому мы должны убедиться, что наш ведомый сервер MySQL сконфигурирован с помощью табличных хранилищ.

```
sudo vim /etc/mysql/my.cnf
```

Добавьте следующие ниже строки в файл `my.cnf`:

```
[mysqld]
master-info-repository    = TABLE
relay-log-info-repository = TABLE
```

Вы можете проверить, активировано ли табличное хранилище, посредством командной оболочки MySQL:

```
mysql> SHOW VARIABLES LIKE '%repository%';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| master_info_repository    | TABLE |
| relay_log_info_repository | TABLE |
+---------------------------+-------+
```

Теперь необходимо изменить конфигурационные файлы для `server-id`, чтобы убедиться, что все серверы будут иметь уникальный идентификатор сервера, используемый в репликации.

Рассмотрим использование трех серверов для многоисточниковой репликации следующим образом:

- 12.34.56.789: ведущая база данных #1;
- 12.34.56.790: ведущая база данных #2;
- 12.23.34.456: ведомая база данных.

Как мы отметили ранее в разделе «*Репликация с использованием глобальных идентификаторов транзакций*», базовая настройка транзакций на основе GTID будет на серверах Ведущий сервер #1 и Ведущий сервер #2 с помощью `gtid_mode=ON`.

Чтобы гарантировать, что для репликации ведомые серверы используют табличные хранилища, мы можем создать ведомых пользователей на ведущих серверах. Как только конфигурации проверены, мы можем добавить ведущие серверы к ведомому серверу с помощью команды `CHANGE MASTER`.

Например, добавьте новый ведущий сервер с именем хоста Ведущий сервер #1, используя порт `3451`, в канал с именем `master-1`:

```
CHANGE MASTER TO MASTER_HOST='12.34.56.789', MASTER_USER='shabbirslave',
MASTER_PORT=3451,
MASTER_PASSWORD='password',
MASTER_AUTO_POSITION = 1 FOR CHANNEL 'master-1';
```

Таким же образом можно добавить еще один сервер, `master-2`, в репликационный канал:

```
CHANGE MASTER TO MASTER_HOST='12.34.56.790', MASTER_USER='shabbirslave',
MASTER_PORT=3451,
MASTER_PASSWORD='password',
MASTER_AUTO_POSITION = 1 FOR CHANNEL 'master-2';
```

Давайте рассмотрим небольшой пример, чтобы понять, как данные реплицируются с обоих ведущих серверов на ведомые серверы:

```
CREATE DATABASE `hrms`;
use hrms;
CREATE TABLE `employee` (
 `emp_id` int(9) NOT NULL AUTO_INCREMENT,
 `emp_name` varchar(60) NOT NULL,
 `emp_job_title` varchar(60) NOT NULL,
 `emp_joining_date` date NOT NULL,
 PRIMARY KEY (`emp_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1;
```

Ключ к тому, чтобы заставить работать многоисточниковую репликацию, состоит в том, чтобы убедиться, что у вас нет одинаковых первичных ключей на обоих ведущих серверах. Если оба ведущих сервера имеют одинаковый первичный ключ для двух разных записей, данные могут быть повреждены, как только они достигнут ведомого сервера. Давайте создадим ту же базу данных с разными автоинкрементными ключами.

Вы можете чередовать значения для нашего столбца первичного ключевого `employee_id`. Вы можете сделать это в приложении, но простой способ – использовать переменную `auto_increment`. В конфигурационном файле вам следует добавить это для обоих ведущих баз данных:

```
[mysqld]
auto_increment_increment = 2
```

Ведущий сервер #1 – создайте новую базу данных как hrms с автоинкрементом 100000, как показано в следующем ниже фрагменте:

```
CREATE DATABASE `hrms`;
use hrms;
CREATE TABLE `employee` (
 `emp_id` int(9) NOT NULL AUTO_INCREMENT,
 `emp_name` varchar(60) NOT NULL,
 `emp_job_title` varchar(60) NOT NULL,
 `emp_joining_date` date NOT NULL,
 PRIMARY KEY (`emp_id`)
) ENGINE=InnoDB AUTO_INCREMENT=100000;
```

Ведущий сервер #2 – та же самая база данных, которую мы создадим в Ведущем сервере 2 с автоинкрементом 100001:

```
CREATE DATABASE `hrms`;
use hrms;
CREATE TABLE `employee` (
 `emp_id` int(9) NOT NULL AUTO_INCREMENT,
 `emp_name` varchar(60) NOT NULL,
 `emp_job_title` varchar(60) NOT NULL,
 `emp_joining_date` date NOT NULL,
 PRIMARY KEY (`employee_id`)
) ENGINE=InnoDB AUTO_INCREMENT=100001;
```

Предположим, что оба ведущих сервера уже настроены с конфигурацией ведущего GTID. Теперь взгляните на каждый ведущий сервер, используя команду SHOW MASTER STATUS:

```
mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
File: mysql-bin.000001
Position: 154
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set:1
row in set (0.00 sec)
```

Добавим немного данных в Ведущий сервер #1 и посмотрим, что происходит с состоянием ведущего сервера:

```
mysql> INSERT INTO employee (emp_name, emp_job_title,
emp_joining_date,emp_joining_date) VALUES('Shabbir','Lead
Consultant','2015-09-29');
Query OK, 1 row affected (0.02 sec)

mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
File: mysql-bin.000001
Position: 574
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set: 63a7971c-b48c-11e5-87cf-f7b6a723ba3d:1
1 row in set (0.00 sec)
```

```
mysql> INSERT INTO employee (emp_name, emp_job_title,
emp_joining_date,emp_joining_date) VALUES('Jaydip','Sr
Consultant','2014-10-12');
Query OK, 1 row affected (0.02 sec)

mysql> show master status\G
*************************** 1. row ***************************
File: mysql-bin.000001
Position: 994
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set: 63a7971c-b48c-11e5-87cf-f7b6a723ba3d:1-2
1 row in set (0.00 sec)

mysql> select * from employee;
+-------------+--------------+----------------+--------------+
| employee_id | employee_name | job_title       | joining_year |
+-------------+--------------+----------------+--------------+
| 100001      | Jaydip        | Sr Consultant   | 2014-10-12   |
| 100003      | Shabbir       | Lead Consultant | 2015-09-29   |
+-------------+--------------+----------------+--------------+
2 rows in set (0.00 sec)
```

Вы можете увидеть, как значения для столбца `employee_id` теперь увеличиваются на два. Мы уже можем вставить две строки данных в Ведущий сервер #2:

```
mysql> INSERT INTO employee (employee_name, job_title,
joining_date,joining_date) VALUES('Kandarp','Lead
Consultant','2014-12-11');
mysql> INSERT INTO employee (employee_name, job_title,
joining_date,joining_date) VALUES('Chintan','Director','2013-01-01');

mysql> show master status\G
*************************** 1. row ***************************
File: mysql-bin.000005
Position: 974
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set: 75e2e1dc-b48e-11e5-83bb-1438deb0d51e:1-2
1 row in set (0.00 sec)

mysql> select * from employee;
+----------+----------+----------------+-----------------+
| emp_id   | emp_name | emp_job_title   | emp_joining_year |
+----------+----------+----------------+-----------------+
| 100002   | Kandarp  | Lead consultant | 2014-12-11       |
| 100004   | Chintan  | Director        | 2013-01-01       |
+----------+----------+----------------+-----------------+
2 rows in set (0.00 sec)
```

Теперь у вас есть два набора глобальных идентификаторов GTID для репликации на ведомый сервер.

Наборы GTID в Ведущем #1 и Ведущем # 2:

```
63a7971c-b48c-11e5-87cf-f7b6a723ba3d:1-2
75e2e1dc-b48e-11e5-83bb-1438deb0d51e:1-2
```

Теперь давайте посмотрим, как данные реплицируются с обоих серверов. Прежде чем мы начнем репликацию, давайте проверим состояние ведомого сервера:

```
mysql> show slave status\G
Empty set (0.00 sec)
```

Добавим базу данных ведущего сервера в ведомый сервер, используя `CHANGE MASTER`:

```
CHANGE MASTER TO MASTER_HOST='12.34.56.789', MASTER_USER='shabbirslave',
MASTER_PORT=3451,
MASTER_PASSWORD='password',
MASTER_AUTO_POSITION = 1 FOR CHANNEL 'master-1';
```

Теперь мы можем запустить ведомый сервер для канала `master-1`:

```
mysql> START SLAVE FOR CHANNEL 'master-1';
```

Эта команда одновременно запустит `SQL_THREAD` и `IO_THREAD`. Теперь, если вы проверите состояние ведомого сервера, то можно увидеть, что ваши глобальные идентификаторы GTID из Ведущего сервера #1 уже были извлечены и применены к базе данных:

```
mysql> SHOW SLAVE STATUS FOR CHANNEL 'master-1'\G
*************************** 1. row ***************************
Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.1.142
...
Master_UUID: 63a7971c-b48c-11e5-87cf-f7b6a723ba3d
...
Slave_SQL_Running_State: Slave has read all relay log; waiting for more
updates
...
Retrieved_Gtid_Set: 63a7971c-b48c-11e5-87cf-f7b6a723ba3d:1-2
Executed_Gtid_Set: 63a7971c-b48c-11e5-87cf-f7b6a723ba3d:1-2
Auto_Position: 1
...
Channel_Name: master-1
```

Если вы посмотрите на таблицу `employee` в ведомом сервере, вы сможете найти обе записи из Ведущего сервера #1.

Теперь давайте запустим репликацию для второго ведущего сервера:

```
CHANGE MASTER TO MASTER_HOST='12.34.56.790', MASTER_USER='shabbirslave',
MASTER_PORT=3451,
MASTER_PASSWORD='password',
MASTER_AUTO_POSITION = 1 FOR CHANNEL 'master-2';
```

Теперь, если мы проверим состояние ведомого сервера снова, то все глобальные идентификаторы GTID из обоих ведущих серверов будут извлечены и применены к ведомой базе данных.

Репликация на основе инструкций против репликации на основе строк

В репликации на основе инструкций ведущий сервер MySQL записывает события как операторы SQL в файле двоичного журнала `binlog`. Эти инструкции подхва-

тываются ведомыми серверами MySQL и воспроизводятся таким же образом, как они были выполнены на ведущем сервере.

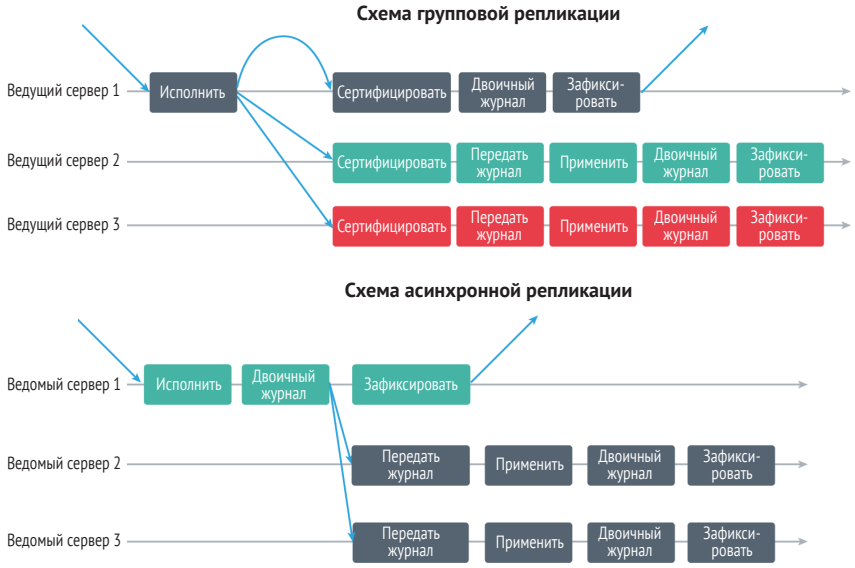
В репликации на основе строк ведущий сервер MySQL записывает события как фактические строки, которые указывают на то, как строки изменяются на ведущем сервере.

Групповая репликация

Из традиционной репликации MySQL вытекает архитектура «ведущий-ведомый», когда операции записи выполняются одним сервером, действующим как ведущее устройство, а другие серверы, действующие как ведомые устройства, сконфигурированы периодически выполнять репликацию, чтобы извлечь данные из ведущего устройства и сохранить локально. Каждое ведомое устройство в репликации «ведущий-ведомый» имеет свою собственную копию данных, и ведомые устройства не делятся какой-либо информацией между собой. В сущности, мы можем назвать этот механизм неразделяемым.

MySQL 8 имеет встроенную поддержку групповой репликации. Групповая репликация подразумевает наличие нескольких серверов, каждый из которых может выступать в качестве ведущего сервера. Серверы в группе координируются при помощи механизма передачи сообщений. Например, если запрашиваются какие-либо операции фиксации транзакций, этот запрос передается серверной группе. Затем серверная группа утверждает транзакцию, и один из серверов фиксирует операцию записи, которая затем реплицируется в группу. Для любой операции чтения утверждение из группы серверов не требуется, и все такие запросы на чтение выполняются немедленно. Эта архитектура немного сложней, чем универсальная архитектура «ведущий-ведомый», но она обеспечивает лучшую атомарность и последовательность.

На следующем ниже рисунке показано, как работает групповая репликация и чем она отличается от традиционной репликации:



Предварительные условия для групповой репликации

При выполнении групповой репликации необходимо соблюдать определенные требования. Приведем эти общие требования.

Данные должны храниться в подсистеме хранения данных InnoDB: поскольку групповая репликация MySQL полностью управляема транзакциями, может существовать возможность одновременного выполнения транзакций на многочисленных серверах. Поэтому, если во время выполнения возникает какой-либо конфликт, MySQL сделает откат к предыдущему безопасному состоянию. Таким образом, подсистема хранения данных, которая поддерживает функционал фиксации и отката транзакций, может использоваться только для групповой репликации. Прямо сейчас в MySQL 8 транзакции поддерживаются лишь подсистемой InnoDB.

Каждая таблица должна иметь первичный ключ: поскольку MySQL использует первичный ключ для разрешения любого конфликта, который возникает при параллельном выполнении транзакций в более чем одном сервере в группе, в каждой таблице в базе данных, которую мы используем для репликации, должен быть явным образом определен первичный ключ.

Сеть IPv4: групповой коммуникационный механизм, используемый групповой репликацией MySQL, поддерживает только IPv4.

Низкая задержка, высокая пропускная способность сети: если серверы, участвующие в групповых репликациях, удалены друг от друга, то задержка фиксации или отката транзакций будет высокой, поэтому желательно, чтобы в групповой репликации серверы находились достаточно близко друг к другу.

До девяти серверов в группе: размер группы конструктивно ограничен и поддерживает максимум до девяти серверов.

Конфигурирование групповой репликации

Откройте главный конфигурационный файл MySQL на каждом сервере MySQL в группе:

```
sudo vim /etc/mysql/my.cnf
```

Перейдите к разделу `[mysqld]` файла `my.cnf` и пошагово добавьте конфигурацию:

```
[mysqld]
gtid_mode = ON
enforce_gtid_consistency = ON
master_info_repository = TABLE
relay_log_info_repository = TABLE
binlog_checksum = NONE
log_slave_updates = ON
log_bin = binlog
binlog_format = ROW
transaction_write_set_extraction = XXHASH64
loose-group_replication_bootstrap_group = OFF
loose-group_replication_start_on_boot = OFF
loose-group_replication_ssl_mode = REQUIRED
loose-group_replication_recovery_use_ssl = 1# General replication settings
```

В этом разделе содержатся общие параметры, относящиеся к глобальным идентификаторам транзакций и ведению двоичного журнала, которые требуются для групповой репликации, а также конфигурируется SSL для группы.

Параметры групповой репликации

Чтобы настроить групповую репликацию на всех членах группы, нам нужно настроить несколько параметров. Нужно настроить уникальный идентификатор группы, используя `loose-group_replication_group_name`. Это будет уникальный идентификатор группы. Нужно добавить в белый список всех членов группы, используя параметр `loose-group_replication_ip_whitelist`. Также нужно предоставить параметр `seed loose-group_replication_group_seeds`, который идентичен параметру `whitelist`, только с другим начальным числом. Он содержит номер порта с IP-адресами серверов. Ниже приведен параметр конфигурации, который необходимо настроить.

```
loose-group_replication_group_name = "959cf631-538c-415d-8164-ca00181be227"
loose-group_replication_ip_whitelist =
"213.121.111.112,213.121.123.2,213.121.121.324"
loose-group_replication_group_seeds =
"213.121.111.112:33061,213.121.123.2:33061,213.121.121.324:33061"
```

Выбор между единственным ведущим или многочисленными ведущим серверами

Конфигурация с одним ведущим сервером имеет только одного члена группы, который выполняет операции записи, в то время как групповая репликация с многочисленными ведущими серверами позволяет всем членам группы выполнять операции записи. Ниже приведена конфигурация, которая должна быть активирована, для того чтобы разрешить групповую репликацию с многочисленными ведущими серверами.

```
loose-group_replication_single_primary_mode = OFF
loose-group_replication_enforce_update_everywhere_checks = ON
```

Эти параметры настройки должны быть одинаковыми на каждом из ваших серверов MySQL.

Можно изменить конфигурацию групповой репликации с многочисленными ведущими серверами на конфигурацию с одним ведущим сервером или наоборот, не теряя никаких данных. Для изменения групповой конфигурации необходимо изменить конфигурацию каждого сервера в группе. После изменения конфигурации требуется перезапуск службы MySQL на каждом члене группы. Для настройки групповой репликации с новыми параметрами требуется простой в нерабочем состоянии.

Специфические для хоста параметры конфигурации

Определенные конфигурации специфичны для каждого сервера в группе, который также должен быть настроен.

- ID сервера;
- локальный адрес сервера;
- удаленный адрес для совместного использования с другим участником;
- локальный адрес и номер порта сервера.

`server_id` – это уникальный номер для каждого члена групповой репликации. Можно назначить любое уникальное значение для каждого члена группы. В нашем примере для первого члена просто назначьте число 1 и увеличивайте это число каждого дополнительного хоста. `bind-address` и `report_host` – это IP-адрес

сервера, который будет использоваться для подключения внешних клиентов. `loose-group_replication_local_address` также является IP-адресом сервера с номером порта (33061), добавленным в конец IP-адреса:

```
server_id = 1
bind-address = "213.121.111.112"
report_host = "213.121.111.112"
loose-group_replication_local_address = "213.121.111.112:33061"
```

Примените эту конфигурацию к каждому серверу MySQL и сохраните файл `my.cnf`.

На этом шаге мы готовы к активации групповой репликации. Все настройки начальной загрузки были сделаны. Теперь мы можем перезапустить службу MySQL на каждом члене группы.

```
sudo service mysql restart
```

По умолчанию для любого запроса на подключение, поступающего на сервер MySQL, СУБД MySQL использует порт 3306. Поэтому, если на вашем сервере установлен какой-либо брандмауэр, примените приведенные ниже команды, чтобы добавить их в список разрешенных.

```
sudo ufw allow 33061
sudo ufw allow 3306
```

С доступом к открытым портам MySQL мы можем создать пользователя репликации и активировать плагин групповой репликации.

Конфигурирование пользователя репликации и активация плагина групповой репликации

Теперь наши серверы готовы к групповой репликации. Поэтому, прежде чем мы настроим плагин групповой репликации, мы должны создать отдельного пользователя для цели репликации. Пожалуйста, имейте в виду, что при создании пользователя для репликации двоичный журнал должен быть отключен, чтобы избежать конфликта, если начинается процесс репликации и серверы начинают читать двоичные журналы для репликации.

Мы предоставим пользователю для репликации привилегии репликации, а затем очистим привилегии для реализации изменений. Теперь мы можем снова активировать двоичные журналы для возобновления нормальной работы.

При создании пользователя репликации обязательно используйте безопасный пароль:

```
SET SQL_LOG_BIN=0;
CREATE USER 'shabbir_user'@'%' IDENTIFIED BY 'password' REQUIRE SSL;
GRANT REPLICATION SLAVE ON *.* TO 'shabbir_user'@'%';
FLUSH PRIVILEGES;
SET SQL_LOG_BIN=1;
```

Используйте инструкцию `CHANGE MASTER TO`, чтобы сконфигурировать сервер для использования заданных учетных данных для каналов репликации `group_replication_recovery`, в последующем это потребуется для восстановления своего состояния из другого члена репликации:

```
CHANGE MASTER TO MASTER_USER='shabbir_user',
MASTER_PASSWORD='password' FOR CHANNEL 'group_replication_recovery';
```

Плагин репликации `Group` требуется в MySQL 8, чтобы запустить групповую репликацию. Этот плагин может быть установлен с помощью следующей ниже команды:

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

Следующая далее команда покажет, был требуемый плагин активирован или нет:

```
SHOW PLUGINS;
```

Запуск групповой репликации

Мы выполнили конфигурацию на разных членах группы. Мы создали отдельных пользователей репликации, и мы активировали плагин групповой репликации. Теперь мы можем настроить базу данных для запуска репликации.

Узел начальной загрузки

Следующие ниже шаги необходимо выполнить на **одиночном члене группы**:

```
SET GLOBAL group_replication_bootstrap_group=ON;
START GROUP_REPLICATION;
SET GLOBAL group_replication_bootstrap_group=OFF;
```

Далее создайте тестовую базу данных и таблицу, чтобы протестировать репликацию:

```
CREATE DATABASE `hrms`;
use hrms;
CREATE TABLE `employee` (
 `emp_id` int(9) NOT NULL AUTO_INCREMENT,
 `emp_name` varchar(60) NOT NULL,
 `emp_job_title` varchar(60) NOT NULL,
 `emp_joining_date` date NOT NULL,
PRIMARY KEY (`emp_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1;

INSERT INTO employee (emp_name, emp_job_title,
emp_joining_date,emp_joining_date) VALUES('Shabbir','Lead
Consultant','2015-09-30');
```

Теперь запустите групповую репликацию на **втором сервере**. Поскольку у нас уже есть активный член репликации, нам не нужно выполнять начальной загрузки группы, и мы можем просто присоединить его к ней:

```
START GROUP_REPLICATION;
```

Таким же образом можно запустить еще один сервер. Чтобы проверить список членов из `replication_group_members`, выполните следующие ниже действия:

```
SELECT * FROM performance_schema.replication_group_members;
Output:
+--------------+-----------+-------------+-------------+--------------+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
```

```
+-------------+----------+------------+------------+-------------+
| group_replication_applier | 13324ab7-1b01-11e7-9dd1-22b78adaa992 |
213.121.111.112 | 3306 | ONLINE |
| group_replication_applier | 1ae4b211-1b01-11e7-9d89-ceb93e1d5494 |
213.121.123.2 | 3306 | ONLINE |
| group_replication_applier | 157b597a-1b01-11e7-9d83-566a6de6dfef |
213.121.121.324 | 3306 | ONLINE |
+-------------+----------+------------+------------+-------------+
3 rows in set (0.01 sec)
```

Таким образом вы можете добавить многочисленные экземпляры MySQL, когда захотите.

Резюме

В этой главе вы познакомились с высокой доступностью и узнали, зачем она требуется. Затем мы рассмотрели различные методы репликации для достижения высокой доступности в архитектурах и подходах MySQL 8. Мы разобрались с различными типами методов репликации в MySQL 8 и удобством использования каждого метода. Мы также изучили, как настроить каждый тип репликации с несколькими примерами в реальном режиме времени.

Теперь пришло время перейти к следующей главе, где мы увидим замечательные методы достижения производительности, масштабируемости, безопасности и доступности с помощью MySQL 8.

Глава 7

Практические рекомендации по работе с MySQL 8

В предыдущей главе вы познакомились с тем, как использовать репликацию для создания высокодоступных решений. Она охватывала целый ряд интересных аспектов, таких как аварийное переключение, групповые репликации, кластеризация и т. д. В этой главе мы рассмотрим практические рекомендации по работе с MySQL 8, долгожданный раздел, который обещает учесть многие недостатки предыдущих версий и имеет новые захватывающие особенности. MySQL 8 обещает быть не просто автономной системой управления базами данных, но и будет играть значительную роль в различных областях, включая решения для больших данных.

Темы, которые мы рассмотрим в этих главах, приведены ниже:

- сравнительные испытания и конфигурации MySQL;
- рекомендации в отношении запросов MySQL;
- рекомендации в отношении конфигурации Memcached;
- рекомендации в отношении репликации.

Благодаря значительным оптимизациям и изменениям версия MySQL 5.7 была продвинута прямо до версии MySQL 8. В MySQL 8 сняты ограничения на количество файлов, ранее ограничивавших количество баз данных, которые можно было иметь. Есть и другие подобные захватывающие особенности, которые мы рассмотрели в главе 1 «*Введение в большие данные и MySQL 8*». СУБД MySQL 8 теперь может хранить в базе данных миллионы таблиц. И при этом она очень быстро вносит изменения в таблицы.

Эта глава непременно заслуживает особого внимания, поскольку знание лучших практических приемов MySQL 8 не только влияет на производительность базы данных, масштабируемость, безопасность и доступность, но и в целом определяет то, насколько ваша система производительна для конечного пользователя. Ведь наша конечная цель именно в этом, не правда ли? Давайте посмотрим на результаты сравнительных испытаний, полученные в нашей тестовой лаборатории, которые наверняка вас удивят!

В этой главе будут рассмотрены следующие темы:

- сравнительные испытания и конфигурации MySQL;
- рекомендации в отношении запросов MySQL;
- рекомендации в отношении конфигурации Memcached;
- рекомендации в отношении репликации.

Сравнительные испытания и конфигурации MySQL

Мы прошлись по различным новым функциональным возможностям и улучшениям MySQL 8. И это нас воодушевляет, так как производительность всегда стоит на первом месте. Во время написания настоящей книги компания Oracle еще не опубликовала результаты испытаний MySQL 8. Мы не стали ждать и выполнили этот анализ самостоятельно по нескольким направлениям.

Рекомендации в отношении конфигурации MySQL являются вишенкой на торте, а без вишенки торт выглядит незаконченным. В дополнение к конфигурациям сравнительные испытания помогают нам проверять и находить узкие места и устранять их. Давайте рассмотрим несколько конкретных областей, которые помогут нам понять лучшие приемы настройки и увеличения производительности.

Использование ресурсов

Активность ввода-вывода, использование процессора и памяти – это то, что вы не должны пропустить. Эти метрические показатели помогают нам узнать, как работает система при выполнении сравнительных испытаний и во время масштабирования. Они также помогают вывести показатель влияния в расчете на транзакцию.

Увеличение длительности нагрузочных тестов

Мы часто хотели бы оперативно взглянуть на метрики производительности, однако обеспечение одинакового поведения MySQL на более продолжительном промежутке тестирования также является ключевым элементом. Когда вы увеличиваете длительность нагрузочных тестов, есть несколько основных моментов, которые могут повлиять на производительность, такие как фрагментация памя-

ти, ухудшение ввода-вывода, влияние после накопления данных, управление кешем и т. д. Мы не хотим перезапускать нашу базу данных, только чтобы очистить от ненужных элементов, правильно? Следовательно, для проверки стабильности и производительности предлагается проводить сравнительный анализ в течение длительного времени.

Репликация параметров производственной среды

Давайте проведем сравнительное испытание в производственно-репликационной среде. Постойте. Давайте отключим репликацию базы данных в репликационной среде, пока мы не закончим со сравнительным испытанием. В яблочко! Мы получили хорошие цифры!

Часто бывает так, что мы не симулируем полностью то, что собираемся настроить в производственной среде. В конечном счете это может обойтись дорого, поскольку мы непреднамеренно будем проводить сравнительное испытание в условиях, которые могут оказать негативное влияние, когда они будут в производственной среде. Во время выполнения сравнительного испытания реплицируйте производственные параметры, данные, рабочую нагрузку и т. п. в реплицированной среде.

Сопоставимость пропускной способности и задержки

Пропускная способность и задержка идут рука об руку. Важно, чтобы ваши глаза в первую очередь были сосредоточены на пропускной способности, однако с течением времени стоит переключиться и на задержку. Провалы производительности, медлительность или потери скорости отмечались в InnoDB еще с ранних времен. К настоящему времени эта подсистема стала значительно совершеннее, но поскольку в зависимости от вашей рабочей нагрузки могут быть и другие случаи, всегда хорошо следить за пропускной способностью вместе с задержкой.

Sysbench может сделать больше

Sysbench – это замечательный инструмент для имитации ваших рабочих нагрузок, будь то тысячи таблиц, интенсивные транзакции, данные в памяти и т. п. Это великолепный инструмент для имитации, который обеспечит вас приличной информацией.

Мир виртуализации

Хотелось бы оставить все простым; голый металл в сравнении с виртуализацией не то же самое. Следовательно, при выполнении сравнительных испытаний измеряйте свои ресурсы в соответствии с вашей средой. Вы будете удивлены, когда увидите разницу в результатах, если вы будете сравнивать оба компонента.

Параллелизм

Большие данные опираются на тяжелую рабочую нагрузку за счет объема данных; обеспечение высокого параллелизма играет важную роль. MySQL 8 расширяет поддержку максимального количества ядер процессоров в каждом новом выпуске. Следует позаботиться об оптимизации параллелизма на основе своих потребностей и аппаратных ресурсов.

Фоновая нагрузка

Не пропускайте факторы, которые работают в фоновом режиме, такие как отчетность для аналитической обработки больших данных, резервные копии и операции на лету во время проведения сравнительного испытания. Влияние таких скрытых нагрузок или рудиментарных контрольных рабочих нагрузок может огорчить вас на многие дни (и ночи).

Суть вашего запроса

Ой, мы, кажется, пропустили оптимизатор? Пока нет. **Оптимизатор** – это мощный инструмент, который будет читать суть вашего запроса и давать рекомендации. Этот инструмент мы используем перед внесением изменений в запрос в рабочей среде. Он просто незаменим, когда имеются сложные запросы, которые нужно оптимизировать.

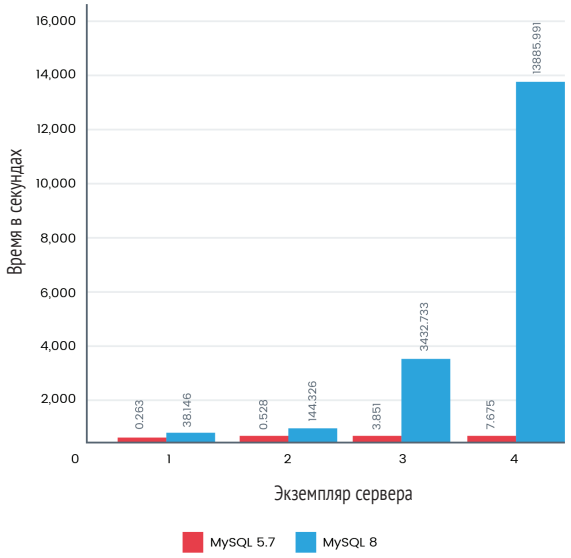
Вот несколько областей, на которые мы должны обратить внимание. Давайте теперь посмотрим на несколько сравнительных испытаний MySQL 8, по сравнению с MySQL 5.7.

Сравнительные испытания

Для начала давайте выберем все имена столбцов из всех таблиц InnoDB. Ниже приведен запрос, который мы выполнили:

```
SELECT t.table_schema, t.table_name, c.column_name
FROM information_schema.tables t,
information_schema.columns c
WHERE t.table_schema = c.table_schema
AND t.table_name = c.table_name
AND t.engine='InnoDB';
```

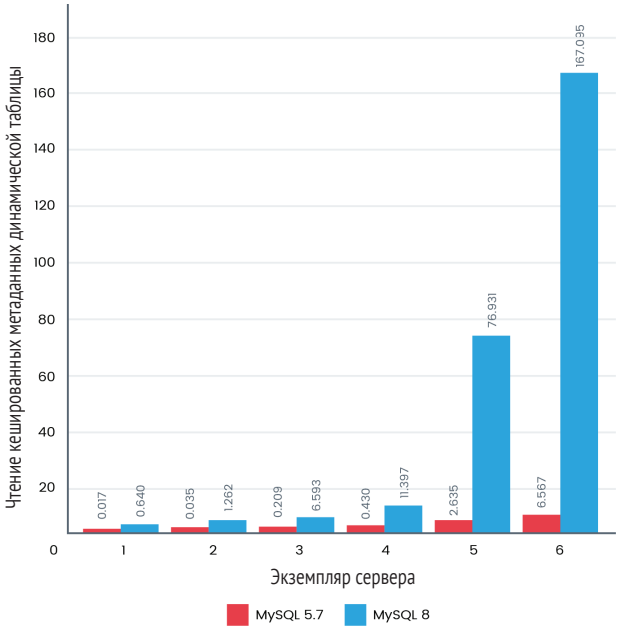
Следующий ниже рисунок показывает, что СУБД MySQL 8, имея четыре экземпляра, работала в тысячу раз быстрее:



После этого мы также выполнили сравнительный тест по поиску метаданных статической таблицы. Ниже приведен запрос, который мы выполнили:

```
SELECT TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE, ENGINE, ROW_FORMAT
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'chintan%';
```

На следующем ниже рисунке показано, что MySQL 8 выполняется примерно в 30 раз быстрее, чем MySQL 5.7:

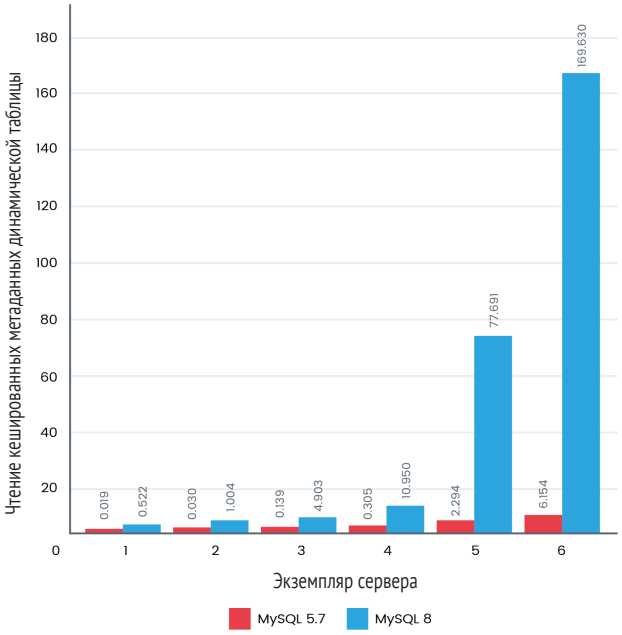


Это заставило нас немного углубиться в детали. Поэтому мы решили выполнить последний сравнительный тест, чтобы найти метаданные динамической таблицы.

Ниже приведен запрос, который мы выполнили:

```
SELECT TABLE_ROWS
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'chintan%';
```

На следующем ниже рисунке показано, что MySQL 8 выполняется примерно в 30 раз быстрее, чем MySQL 5.7:



MySQL 8.0 приносит огромное улучшение производительности в работе с таблицей. Масштабирование до миллиона таблиц, что является необходимостью для многих потребностей, связанных с большими данными, теперь достижимо. Мы с нетерпением ждем множества официальных сравнительных тестов, как только СУБД MySQL 8 будет доступна для общецелевого применения.

Давайте теперь рассмотрим нашу следующую тему, которая облегчит вашу жизнь – все, что следует принять во внимание при формировании запроса.

Рекомендации в отношении вопросов MySQL

Трудно составить рекомендации в отношении запросов для справки и повторного использования. Они всегда будут разными в зависимости от характера вашего приложения, архитектуры, конструкции, структуры таблицы и т. д. Однако при написании запросов MySQL можно принять некоторые меры предосторожности с целью повышения производительности, масштабируемости и целостности.

Давайте рассмотрим некоторые из лучших практических приемов, которые мы должны иметь в виду при разработке или написании запросов MySQL.

Типы данных

Таблица базы данных состоит из нескольких столбцов, имеющих типы данных числовой или строковый. MySQL 8 предоставляет различные типы данных, не ограничиваясь просто числовым или строковым.

- Чем меньше, тем лучше. Поскольку MySQL загружает данные в память, большой объем данных отрицательно повлияет на производительность. Более мелкие наборы могут вместить больше данных в памяти и снизить накладные расходы на использование ресурсов.

- Зафиксируйте длину. Если не зафиксировать длину типа данных, ему придется каждый раз извлекать требующуюся информацию. Поэтому везде, где это возможно, лучше ограничить длину данных с помощью типа данных `char`.

Not null

Ненулевые данные Not null – это то, что MySQL не нравится. Ненулевые столбцы используют больше памяти, влияют на производительность и требуют дополнительной обработки в MySQL. Оптимизация запросов, ссылающихся на нулевые данные, также представляет свои трудности. Когда индексируется столбец данных null, он использует дополнительные байты для каждого входа.

Индексация

Индексирование играет важную роль, так как оно улучшает производительность плохо спроектированного запроса и структуры таблицы или может даже превратить хорошо спроектированный запрос в плохой, что повлияет на производительность.

Индекс поисковых полей

Как правило, мы индексируем поля, которые в запросах MySQL используются в качестве фильтров. Это, очевидно, помогает быстрее читать, но может негативно повлиять на операции записи/обновления, поэтому индексирование только того, что вам нужно, будет разумным решением.

Типы данных и соединения

MySQL может выполнять соединения для разных типов данных, но производительность может упасть, если СУБД MySQL будет поручено использовать для полей различные типы данных, и ей придется конвертировать из одного типа в другой для каждой строки.

Составной индекс

Если запрос должен ссылаться на несколько столбцов таблицы, то для таких столбцов может оказаться полезным составной индекс. Составной индекс ссылается на столбцы результатов, заданных первым столбцом, вторым столбцом и т. д.

На производительности запроса сильно сказывается порядок следования столбцов, поэтому при разработке структуры таблицы и индекса необходимо использовать его эффективно.

Укорачивайте первичные ключи

Принцип «чем меньше, тем лучше» подходит и для первичных ключей. Сокращение первичных ключей принесет пользу аналогично тому, как мы обсуждали типы данных. Благодаря меньшему размеру первичных ключей размер индекса будет меньше, и, следовательно, использование кеша будет меньше, поэтому он может вместить больше данных в памяти. Для достижения цели сокращения первичных ключей рекомендуется использовать числовые типы, поскольку они будут намного меньше символов. Этот прием окажется полезным при выполнении соединений, поскольку, как правило, первичные ключи относятся к операции соединения.

Индексируйте все, что можно

Индексирование всего – хорошая идея, но MySQL ее не использует. Знаете ли вы, что СУБД MySQL будет выполнять полное сканирование таблицы, если она должна просканировать свыше 30% индекса? Не индексируйте значения, которые не нужно индексировать.

Мы должны иметь в виду, что индексирование помогает – при правильном подходе – в получении данных; однако при записи/обновлении данных оно является накладными расходами.

Извлекайте все данные

`select *...` – бррр! Не делайте этого, пока нет необходимости. По собственному опыту, до сих пор необходимости в этом не было. Доставка всех данных замедлит время выполнения и в большой степени скажется на использовании ресурсов сервера MySQL. Необходимо предоставить конкретное имя столбца или соответствующие условия.

Приложение сделает работу

Пусть приложение также выполняет работу для MySQL. Вы можете избежать таких предложений, как `ORDER`, предоставив упорядочивание выполнять приложениям. Упорядочивание в MySQL намного медленнее, чем в приложениях. Вы можете идентифицировать запросы, которые должны обрабатываться приложением согласно плану.

Существование данных

Проверка существования данных с помощью предложения `EXISTS` выполняется гораздо быстрее. Предложение `EXISTS` возвращает результат, как только оно получает первую строку из извлеченных данных.

Ограничивайте себя

Ограничьте себя данными, которые необходимо извлечь. Всегда проверяйте, что при извлечении данных используете соответствующие ограничения, поскольку нежелательные данные не будут полезны и повлияют на производительность. В SQL-запросах используйте предложение `LIMIT`.

Анализируйте медленные запросы

Эту практическую рекомендацию следует выполнять всегда. Вы можете забыть про оптимизацию или реализацию запросов, получив в итоге негативное воздействие на производительность по мере роста данных. У вас могут возникнуть изменения в потребностях относительно данных, которые должны извлекаться как раз в том месте, где вы забыли проанализировать воздействие запросов. Неплохо всегда следить за медленными запросами, которые можно сконфигурировать в MySQL и их оптимизировать.

Стоимость запроса

Какова стоимость вашего запроса? Объяснить – значит получить правильный ответ на этот вопрос. Используйте предложение `EXPLAIN QUERY`, чтобы узнать, что

влияет на запрос – будь то полное сканирование таблицы, сканирование индекса, доступ к диапазону и т. д. Мудро используйте информацию, предоставленную предложением `EXPLAIN`, чтобы оптимизировать запрос дальше. Это замечательный, быстрый, удобный инструмент MySQL. Если вы знаете, что вы сделали все возможное, индексирование придет как спасение для оптимизации его дальше с учетом ваших потребностей.

Самая лучшая практическая рекомендация при написании запроса начинается с определения потребностей, проектирования, реализации и текущего сопровождения. Это полный жизненный цикл, который невозможно разнообразить. Понимание схем, индексов и результатов анализа играет важную роль. Для нас важны время отклика и оптимальное использование ресурсов.

Мир связей – лично я люблю погружаться в это гораздо глубже, чем мы можем здесь! Запрос будет соответствовать строке или столбцу таблицы или соединяться с другой таблицей. Более того, если у вас не получилось сделать все правильно, вы попробуете отыскать связь из подмножества, которая не требуется. Как же можно забыть про индексы, которые приходят на помощь, если они используются надлежащим образом? Все это вместе покажет наши связи и оперативно ответит на запрос.

Давайте двигаться вперед и обратимся к подробному анализу практических рекомендаций в отношении конфигурации Memcached.

Рекомендации в отношении конфигурации Memcached

В главе 4 «*Использование Memcached в MySQL 8*» мы уже рассмотрели настройку и конфигурирование плагина Memcached вместе с его использованием с различными API. Сейчас же мы пройдемся по лучшим практическим приемам конфигурирования Memcached.

Распределение ресурсов

Память для Memcached не должна выделяться сверх доступной физической памяти или без учета других ресурсов, которые могут использовать память. Если мы выделяем излишнюю память, то высоки шансы, что память под Memcached будет выделена из пространства подкачки. Это может привести к задержке при вставке или выборке значений, поскольку пространство подкачки хранится на диске, что медленнее, чем хранение в памяти.

Архитектура операционной системы

С 32-разрядной архитектурой операционной системы нужно быть осторожным. Как известно, существуют ограничения на предоставление ресурсов в 32-разрядной архитектуре операционной системы. Точно так же и с Memcached – при 4 Гб оперативной памяти с 32-разрядной архитектурой операционной системы ему не должно быть назначено более 3.5 Гб оперативной памяти, поскольку это может привести к ухудшению производительности и сбоям.

Конфигурации по умолчанию

Некоторые принятые по умолчанию ключевые параметры конфигурации всегда должны быть точно настроены в соответствии с вашими потребностями.

- **Выделение памяти**: по умолчанию 64 Мб; этот объем должен быть перенастроен на основе потребностей и тестирования.
- **Соединения**: по умолчанию 1024 одновременных соединения; это количество должно быть изменено на основе потребностей и тестирования.
- **Порт**: по умолчанию прослушивается порт 11211; в целях безопасности должен прослушиваться другой порт.
- **Сетевой интерфейс**: по умолчанию принимаются подключения от всех сетевых интерфейсов; в целях безопасности это должно быть ограничено.

Максимальный размер объекта

Следует рассмотреть возможность конфигурирования максимального размера объекта, который по умолчанию равен 1 Мб. Однако его можно поднять до 128 Мб. Основываясь исключительно на том, какой тип данных вы собираетесь хранить, и, соответственно, максимальный размер объекта должен быть разрешен. Хранение лишних данных в Memcached может отрицательно повлиять на производительность, поскольку будет извлекаться гораздо больше данных.

Ограничение очереди незавершенных заданий

Ограничение очереди незавершенных заданий касается количества соединений с Memcached, которые следует держать в очереди, пока она не достигает предела разрешенных соединений. В идеале количество разрешенных соединений должно быть настроено таким образом, чтобы его было достаточно для большинства ваших потребностей. Ограничение очереди незавершенных заданий может быть полезно, когда существует неожиданная пиковая нагрузка на Memcached. В идеале она не должна выходить за пределы 20% от общего числа соединений. В противном случае это может повлиять на работу системы, извлекающей информацию из Memcached, вследствие больших задержек.

Поддержка больших страниц

В системах с поддержкой больших страниц памяти вам следует разрешить Memcached этим воспользоваться. Поддержка больших страниц помогает выделять большой блок данных для хранения, а также за счет этого уменьшать количество кешей пропущенных вызовов.

Конфиденциальные данные

Хранение конфиденциальных данных в Memcached открывает двери для угрозы, поскольку тот, кто имеет доступ к Memcached, может просмотреть конфиденциальную информацию. Очевидно, следует принять меры предосторожности, чтобы ограничить открытость Memcached. Перед тем как сохранять конфиденциальную информацию в Memcached, вы также можете ее зашифровать.

Ограничение открытости

В Memcached не так много встроенного функционала обеспечения безопасности. Одно из средств – предоставление доступа к Memcached в необходимых границах. Если ваш сервер приложений должен общаться с Memcached, разрешайте доступ к Memcached из сервера только с помощью системных межсетевых правил, таких как таблицы IP или подобные методы.

Отказоустойчивость

Memcached не имеет хороших методов отказоустойчивости. Рекомендуется настроить приложение таким образом, чтобы оно отработало отказ на недоступной ноте и восстановило данные в другую инстанцию. Неплохо иметь, по крайней мере, два настроенных Memcached, чтобы избегать аварийного завершения работы из-за недоступности инстанции.

Пространства имен

Вы можете использовать пространства имен, предоставляемые Memcached, которые, в сущности, добавляют префиксы к данным перед их сохранением в Memcached. Это может помочь, когда у вас несколько приложений, которые обращаются к Memcached. Это очень полезно, и, используя некоторые элементарные принципы соглашения об именах, вы можете получить требуемое решение. Если есть данные, которые хранят имена и фамилии, то, соответственно, можно использовать такие префиксы, как FN и LN. Это поможет вам легко идентифицировать и извлекать данные из приложения.

Механизм кеширования

Один из самых простых способов начать использовать кеширование в Memcached – использовать таблицу с двумя столбцами; вы можете задействовать пространства имен, предоставляемые Memcached, где в основном добавляются префиксы. Первые столбцы должны быть первичным ключом, а схема базы данных должна поддерживать адресацию по уникальному идентификатору вместе с уникальными ограничениями. В случае если вы хотите иметь единое значение элемента путем объединения нескольких значений столбцов, следует позаботиться о выборе соответствующих типов данных.

Запросы, имеющие единственный оператор `WHERE`, могут быть легко увязаны с поисками в Memcached при использовании операторов `=` или `IN` в самих запросах. В случаях, когда используется несколько операторов `WHERE` или анализируются сложные операции, такие как `<`, `>`, `LIKE`, `BETWEEN`, Memcached будет испытывать сложности. Рекомендуется выполнять такие сложные операции с использованием традиционных SQL-запросов к базе данных.

Предпочтительно кешировать целиком все объекты в Memcached, вместо того чтобы пытаться кешировать отдельные строки из MySQL 8. Например, для сайта блогов необходимо кешировать весь объект поста блога в Memcached.

Общая статистика Memcached

Чтобы помочь вам лучше разобраться в статистике Memcached, мы предоставим обзор состояния и производительности. Статистика, возвращаемая Memcached, и ее значение показаны в следующей ниже таблице.

Термины, используемые для определения значения по каждой статистике:

- **32u**: 32-разрядное беззнаковое целое;
- **64u**: 64-разрядное беззнаковое целое;
- **32u:32u**: два 32-разрядных беззнаковых целых, разделенных двоеточием;
- **String**: последовательность символов.

Статистика	Тип данных	Описание
pid	32u	ИД процесса экземпляра Memcached
uptime	32u	Время в рабочем состоянии (в секундах) для этого экземпляра Memcached
time	32u	Текущее время (как эпоха)
version	string	Строка версии этого экземпляра
pointer_size	string	Размер указателей для этого хоста, заданный в битах (32 или 64)
rusage_user	32u:32u	Общее время пользователя для этого экземпляра (секунды:микросекунды)
rusage_system	32u:32u	Общее системное время для этого экземпляра (секунды:микросекунды)
curr_items	32u	Текущее количество элементов, хранимых этим экземпляром
total_items	32u	Общее количество элементов, хранимых в течение жизни этого экземпляра
bytes	64u	Текущее количество байтов, используемых сервером для хранения элементов
curr_connections	32u	Текущее количество открытых соединений
total_connections	32u	Общее количество соединений, открытых с тех пор, как сервер был запущен
connection_structures	32u	Количество соединительных структур, выделенных сервером
cmd_get	64u	Общее количество запросов на извлечение (операций get)
cmd_set	64u	Общее количество запросов на сохранение (операций set)
get_hits	64u	Количество ключей, которые были запрошены и найдены существующими
get_misses	64u	Количество элементов, которые были запрошены и не найдены
delete_hits	64u	Количество ключей, которые были удалены и найдены существующими
delete_misses	64u	Количество элементов, которые были удалены и не найдены
incr_hits	64u	Количество ключей, которые были дополнены и найдены существующими
incr_misses	64u	Количество элементов, которые были дополнены и не найдены
decr_hits	64u	Количество ключей, которые были уменьшены и найдены существующими
decr_misses	64u	Количество элементов, которые были уменьшены и не найдены
cas_hits	64u	Количество ключей, которые сравнивались и обменивались и найдены существующими
cas_misses	64u	Количество элементов, которые сравнивались и обменивались и не найдены
cas_badvalue	64u	Количество ключей, которые сравнивались и обменивались, но (исходное) значение сравнения не совпало с предоставленным значением
evictions	64u	Количество допустимых элементов, удаленных из кеша, чтобы высвободить память для новых элементов
bytes_read	64u	Общее количество байт, прочитанных этим сервером из сети
bytes_written	64u	Общее количество байт, отправленных этим сервером в сеть
limit_maxbytes	32u	Количество байт, которое этому серверу разрешено использовать для хранения
threads	32u	Количество запрошенных рабочих потоков
conn_yields	64u	Количество выдач для соединений (связано с опцией -R)

Ссылка: https://dev.mysql.com/doc/refman/8.0/en/ha-memcached-stats-general.html.

Эти несколько полезных параметров следует учитывать при применении рекомендаций относительно Memcached. Теперь самое время двинуться дальше и рассмотреть практические рекомендации в отношении репликации.

Рекомендации в отношении репликации

СУБД MySQL 8 внесла несколько больших усовершенствований в части репликации. Цель MySQL 8 – это масштабируемость, производительность и безопасность с максимальной целостностью данных, что, как ожидается, также изменит правила игры в больших данных.

Пропускная способность в групповой репликации

Групповая репликация в основном имеет дело с фиксацией транзакций, как только большинство членов групповой репликации подтвердили транзакцию, полученную одновременно. Если общее количество операций записи не превышает емкость членов в групповой репликации, это помогает повысить пропускную способность. В случае если емкость не спланирована должным образом, вы заметите запаздывания на затронутых членах, по сравнению с другими членами в группе.

Определение размеров инфраструктуры

Определение размера инфраструктуры является общим фактором успеха для производительности и контрольного списка лучших практических приемов. Если размеры инфраструктуры определены неправильно или распределение узлов групповой репликации неравномерно, это может нарушить фундамент топологии репликации. Каждый компонент должен быть рассмотрен при определении требуемой от них пропускной способности.

Постоянная пропускная способность

Достижение постоянной пропускной способности является хорошим фактором успеха. Что делать, если вы начинаете испытывать нагрузку, которая начинает влиять на остальных членов в групповой репликации? Может оказаться, что ваш ведущий сервер продолжает принимать дополнительную рабочую нагрузку и будет отставать в течение некоторого времени, после чего он может вернуться к приемлемому уровню до того, как исчерпает все ресурсы. Кроме того, вы можете реализовать методологию организации очередей, которая может предотвратить исчерпывание ресурсов и позволит передать рабочие нагрузки только членам, предопределенным на основании емкости.

При рассмотрении методологии организации очередей необходимо иметь в виду, что очереди не должны расти экспоненциально. Это повлияет на конечного пользователя, поскольку данные будут обновляться с запаздыванием. Вместе с тем для достижения постоянной пропускной способности по всей системе необходимо принимать решение на основе технических и деловых потребностей.

Неподходящая нагрузка

По большому счету, групповая репликация предназначена для того, чтобы разрешить обновления из любого члена группы. Откат транзакций на основе перекры-

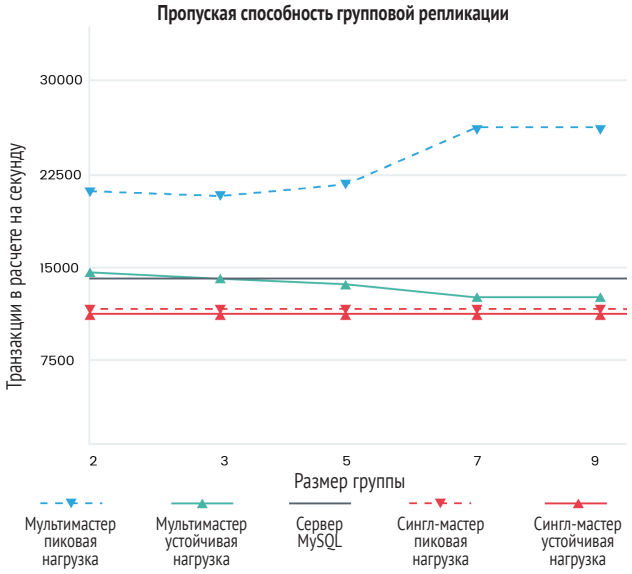
тия строк проверяется для каждой транзакции, остальные фиксируются и отправляются на обновление другим участникам группы. Если несколько обновлений в одной строке происходит часто, это может привести к нескольким откатам. Вы можете столкнуться с циклическими ситуациями, когда один сервер обновляет, запрашивает другие обновления, а другой параллельно уже обновился для той же самой строки, – это приведет к откату.

Чтобы предотвратить такой сценарий, можно дать последнему члену группы применить обновление, после чего вы переходите к другому. Подобные обновления можно направлять только с того же самого узла, где предыдущее было выполнено, чтобы предотвратить вероятность циклического отката.

Масштабируемость операции записи

Распределите рабочую нагрузку записи путем распределения операций записи, что может повысить пропускную способность и масштабируемость при записи. Это будет зависеть от противоречивых рабочих нагрузок, которых вы ожидаете в системе. Полезно распределять нагрузку, когда выполняется пиковая рабочая нагрузка. В обычных случаях, если вы хорошо спланируете емкость с учетом масштабируемости записи, вы увидите тривиальное улучшение.

Взгляните на следующую ниже схему, которая это иллюстрирует:



Вы можете заметить, что с помощью многочисленных ведущих серверов для распределения нагрузки ваша нагрузка имеет лучшую пропускную способность. В конфигурации с многочисленными ведущими серверами также учитывается размер группы.

Резюме

Я уверен, что в процессе чтения главы вы учитывали все, о чем нужно позаботиться или помнить, если в вашей реализации MySQL 8 чего-то не хватает. В этой

главе мы обсудили практические рекомендации по работе с MySQL 8, которые будут полезны на разных стадиях, в том числе на стадиях реализации, использования, управления и поиска неисправностей, и будут заставлять ссылаться на главу «*Практические рекомендации по работе с СУБД MySQL 8*», все зависит от различных примеров использования. Надлежащее тестирование и проверка подтвердят преимущества, получаемые за счет реализации наилучших практических приемов.

Мы широко рассмотрели некоторые захватывающие темы, в том числе сравнительное испытание MySQL 8 и несколько параметров конфигурации. Запросы MySQL также были рассмотрены вместе с практическими рекомендациями по кешированию с использованием Memcached. Наконец, мы обсудили практические рекомендации в отношении репликации MySQL, в которых мы решили несколько критических вопросов. Материала в этой главе, безусловно, недостаточно, но зато он обеспечивает необходимые подсказки.

К настоящему моменту у нас имеется хорошая реализация MySQL 8; пора интегрироваться с другими системами. В следующей главе мы двинемся дальше, чтобы рассмотреть то, как достигается интеграция API NoSQL с решениями для больших данных. Мы пройдемся по различным API, чтобы заставить MySQL и NoSQL обмениваться между собой.

Глава 8

Прикладной программный интерфейс NoSQL для интеграции с решениями для больших данных

В предыдущей главе мы рассмотрели практические рекомендации по работе с MySQL 8, которые мы должны соблюдать в различных аспектах, таких как производительность, реализация, безопасность и т. д. Передовая практика всегда выравнивает продукт с индустриальным стандартом и также дает хороший результат. Мы увидели различные практические приемы для различных этапов, таких как сравнительные испытания и конфигурация, которые детализируют параметры конфигурации, для того чтобы обеспечить более оптимальные результаты сравнительных испытаний. Рекомендации в отношении запросов помогают в разработке сложных запросов и обеспечивают лучшую производительность. Рекомендации в отношении Memcached должны соблюдаться при реализации механизмов кеширования. Правильная конфигурация репликации всегда приведет к лучшей масштабируемости и может обеспечить лучшую производительность с низкой задержкой.

Эта глава посвящена интеграции MySQL 8 для работы как NoSQL с различными API. Мы увидим, как реляционная СУБД MySQL будет работать как NoSQL в целях обработки больших данных. MySQL предоставляет различные API на разных языках программирования, которые мы можем легко интегрировать с приложениями и управлять большими данными. Давайте исследуем некоторые API с элементарными примерами на разных языках программирования.

Ниже приведены темы, которые мы собираемся обсудить в этой главе:

- обзор NoSQL;
- NoSQL против SQL;
- реализация NoSQL API.

Обзор NoSQL

Термин NoSQL – это отсутствие SQL, что означает, что слой SQL пропускается и база данных используется напрямую. Поскольку данные быстро растут каждый

день и большие данные выходят на сцену, чтобы сделать данные масштабируемыми и быстрыми, NoSQL помогает в управлении массивными объемами данных с помощью эволюции **реляционных систем управления базами данных** (реляционных СУБД).

Многие специалисты также интерпретировали термин NoSQL как «**не только SQL**», потому что эта технология представляет собой не полную замену слоя SQL, а, можно сказать, комплементарное дополнение к реляционной базе данных. NoSQL не следует правилам реляционной базы данных и может легко получать доступ к данным в традиционной базе данных.

MySQL уже заслужил свою популярность в структурированном мире, но в соответствии с текущим сценарием данные раздвигают границы, что выводит хранение и анализ на совершенно новый уровень. Этот толчок в развитии актуализировал потребность в NoSQL в условиях больших данных, чтобы выполнять более сильный анализ и обеспечивать эффективную производительность. Таким образом, MySQL разблокировал свой чистый структурированный мир и двинулся вперед к NoSQL наряду с высокой производительностью, высокой доступностью и расширяющимися пределами того, что называется реляционной СУБД.

Давайте посмотрим на основные преимущества наличия базы данных NoSQL.

Быстрое изменение с течением времени

Основная проблема с СУБД – управление изменениями. Поскольку данные растут день ото дня, нам может потребоваться изменить структуру моделей данных в СУБД, что является довольно рискованным и должно делаться тщательным образом, учитывая время простоя в нерабочем состоянии и не затрагивая существующих данных.

NoSQL способна помочь нам справиться с этим быстрым изменением, не затрагивая ничего, потому что NoSQL позволяет приложениям хранить данные и управлять ими с любой структурой, которая нужна, с различными видами хранения, такими как ассоциативное хранилище (ключ-значение) или документоориентированное хранилище.

Масштабирование

Когда из-за больших объемов данных в СУБД в игру вступает масштабирование, нам, возможно, придется побеспокоиться, потому что оно требует дорогостоящих серверов и оборудования для масштабирования СУБД. Вместе с тем новая технология NoSQL предоставляет решения для масштабирования и распределения нагрузки между несколькими хостами с очень низкой стоимостью, по сравнению с СУБД. Она также способна прозрачно выполнять горизонтальное масштабирование по мере увеличения скорости транзакций, не влияя на систему.

Меньше управленческой деятельности

Администрирование базы данных больше не требуется! В случае с СУБД нам может потребоваться администратор для управления базой данных, что действительно является накладными расходами. NoSQL, как правило, спроектирована так, чтобы требовать меньше управленческой деятельности, и обеспечивает автоматическое восстановление, распределение данных и более простые модели данных, а это

позволяет пользователю меньше заниматься управлением, по сравнению с реляционными СУБД. Вместе с тем кто-то всегда должен быть, чтобы обеспечивать производительность, доступность и работу хранилищ данных.

Лучшее для больших данных

Как мы уже отмечали, увеличение скорости транзакций и объема данных породило термин «большие данные». Возможности реляционных СУБД по удовлетворению такой потребности весьма ограничены и также влияют на производительность. NoSQL в основном предназначена для обработки больших данных, которые имеют много возможностей, и ее более простой подход к хранению и извлечению данных действительно повышает производительность.

В настоящее время аналитическая обработка данных является одной из универсальных потребностей большинства предприятий, и NoSQL упрощает аналитическую обработку больших данных. Задача извлечения содержательной деловой аналитической информации из очень крупных объемов данных в условиях традиционных реляционных систем управления баз данных может быть утомительной. Учитывая все факторы, влияющие на непрерывное увеличение данных и динамическую структуру данных, можно сказать, что NoSQL – это самый лучший способ обработки больших данных.

NoSQL против SQL

Давайте рассмотрим основные различия между системами управления базами данных NoSQL и SQL:

NoSQL	SQL
NoSQL – это нереляционная и распределенная система управления базами данных	SQL – это реляционная система управления базами данных
NoSQL масштабируется горизонтально	SQL масштабируется вертикально
NoSQL не использует язык структурированных запросов. У нее имеется язык неструктурированных запросов, который варьируется от СУБД к СУБД	SQL использует язык структурированных запросов для управления данными
NoSQL не имеет фиксированной или предопределенной схемы	SQL имеет предопределенную статическую базу данных
NoSQL хранит данные в парах ключ-значение	SQL хранит данные в табличном формате
NoSQL не подходит для сложных реляционных запросов	SQL лучше всего подходит для сложных реляционных запросов
Предпочтительна для работы с большими данными	Предпочтительна для работы с реляционными данными
NoSQL хранит данные в форме коллекций, где данные могут дублироваться и храниться как единый объект. Следовательно, чтение/запись на едином объекте проще и быстрее	SQL хранит данные нормализованными и разбивает на отдельные таблицы, что позволяет избегать избыточности данных и дублирования. Однако чтение/запись должны выполняться на разных объектах, следовательно, в обработке данных могут иметь место сложные запросы

Реализация API NoSQL

До сих пор мы видели только то, как базы данных NoSQL способны помогать различными способами, а также пробежались по различиям между SQL и NoSQL.

Теперь вам может быть интересно, каким образом мы можем хранить/извлекать данные в MySQL. MySQL 8 уже предоставляет целый ряд API на разных языках программирования для доступа к MySQL, которые работают как NoSQL. В MySQL реализованы интерфейсы NoSQL поверх подсистем хранения данных InnoDB и MySQL Cluster, которые полностью обходят слой SQL.

Следовательно, они сохраняют данные ключ-значение непосредственно в таблицы MySQL с более высокой скоростью, не делая синтаксического анализа и оптимизации SQL. MySQL по-прежнему поддерживает все преимущества своей существующей инфраструктуры реляционной базы данных, и пользователь может продолжить выполнение сложных запросов с помощью SQL на тех же наборах данных.

С MySQL 8 используется в основном два типа NoSQL API на различных языках программирования.

- NoSQL со слоем Memcached
- NoSQL API с Java
- NoSQL API с PHP
- NoSQL API с Python
- NoSQL API с Perl
- NoSQL API с NDB Cluster
- NDB API для NodeJS
- NDB API для Java
- NDB API с C++

Давайте рассмотрим каждый API подробно.

NoSQL со слоем API Memcached

Как мы видели в главе 4 «*Использование Memcached в MySQL 8*», родной Memcached API является частью MySQL 8 и MySQL Cluster. Вы можете использовать этот API для чтения и записи данных с помощью клиента Memcached. В настоящее время многие предприятия для доступа к данным требуют реляционную СУБД на основе SQL с методами NoSQL, и MySQL 8 обеспечивает решение типа «не только SQL» с помощью интерфейса Memcached.

Memcached API непосредственно обращается к подсистемам хранения данных InnoDB и MySQL Cluster без преобразования в SQL наряду с обеспечением низкой задержки и высокой пропускной способности для запросов на чтение/запись. Плагин `daemon_memcached` выполняется на том же пространстве процесса так, что пользователь получает доступ с очень низкой задержкой к своим данным, также задействуя улучшения масштабируемости, предоставляемые подсистемой InnoDB. Обязательно установите и настройте Memcached, как было описано в главе 4 «*Использование Memcached в MySQL 8*». Теперь давайте приступим к разработке клиента Memcached с разными языками программирования.

Давайте предположим, что у нас есть список контактных лиц с простой информацией, которая выглядит следующим образом:

Контактное лицо	Название компании	Адрес электронной почты	Номер мобильного телефона
Jaydip Lakhatariya	KNOWARTH	jaydip.lakhatariya@knowarth.com	9111111111
Chintan Mehta	KNOWARTH	chintan.mehta@knowarth.com	9222222222
Kandarp Patel	KNOWARTH	kandarp.patel@knowarth.com	9333333333
Shabbir Challawala	KNOWARTH	shabiir.challawala@knowarth.com	9444444444

Мы будем хранить эту информацию в СУБД MySQL и обращаться к ней через NoSQL API. Давайте сначала выполним предварительные условия для построения приложения списка контактных лиц, а затем мы увидим NoSQL API на разных языках.

Предварительные условия

Первым шагом в разработке приложения является проектирование базы данных. Нам нужно создать таблицу в MySQL со всеми необходимыми полями. Ниже приведен запрос, который создаст таблицу для контактной информации:

```
CREATE TABLE Contacts (
 id INT PRIMARY KEY,
 firstName VARCHAR(20),
 lastName VARCHAR(20),
 emailAddress VARCHAR(100),
 mobileNumber VARCHAR(10),
 companyName VARCHAR(100)
);
```

Как только таблица была создана успешно, мы должны добавить табличную запись в таблицы Memcached (как было объяснено в главе 4 «*Использование Memcached в MySQL 8*»):

```
INSERT INTO innodb_memcache.containers
SET
 name = "contacts_table",
 db_schema = "demo",
 db_table = "Contacts",
 key_columns = "id",
 value_columns ="firstName,lastName,emailAddress,mobileNumber,companyName"
 unique_idx_name_on_key = "PRIMARY";
```

Это все в рамках предварительных условий. Теперь все готово к тому, чтобы познакомиться с NoSQL API с различными языками программирования.

NoSQL API с Java

Java является наиболее используемым языком программирования, и Memcached предоставляет встроенную библиотеку с пакетом `com.whalin.MemCached`, который легко интегрируется с приложением на Java. Становится очень легко создавать экземпляры и выполнять различные операции, такие как set, get, delete и многие другие. Следующий ниже пример класса Java в первую очередь инициализирует соединение с сервером Memcached в конструкторе и имеет другой метод для различных операций:

```
import com.whalin.MemCached.MemCachedClient;
import com.whalin.MemCached.SockIOPool;
import java.util.Map;

public class ContactMemcachedClient {

  private static String[] servers = {"localhost:11211"};
  private static Integer[] weights = { 1 };
  private MemCachedClient memcachedClient;
  public static final String NAMESPACE = "@@contacts_table";
```

```
/**
 * Инициализировать соединение
 */
public ContactMemcachedClient() {
  SockIOPool pool = SockIOPool.getInstance();
  pool.setServers(servers);
  pool.setWeights( weights );
  if (!pool.isInitialized()) {
    pool.setInitConn(5);
    pool.setMinConn(5);
    pool.setMaxConn(250);
    pool.initialize();
  }
  MemCachedClient memCachedClient = new MemCachedClient();
  this.memcachedClient = memCachedClient;
}

/**
 * Добавить/Обновить значение заданного ключа
 * @param key
 * @param value
 */
public void set(String key, Object value) {
  memcachedClient.set(key, value);
}

/**
 * извлечь значение по ключу
 * @param key
 * @return
 */
public Object get(String key) {
  return memcachedClient.get(key);
}

/**
 * Удалить значение по ключу
 * @param key
 * @return
 */
public Object delete(String key) {
  return memcachedClient.delete(key);
}

/**
 * Извлечь значения как словарь из многочисленных ключей
 * @param keys
 * @return
*/
public Map<String, Object> getAllByKey(String[] keys) {
  return memcachedClient.getMulti(keys);
}

/**
 * Проверить, существует ли значение по заданному ключу или нет
```

```
   * @param key
   * @return
  */
  public boolean isKeyExists(String key) {
    return memcachedClient.keyExists(key);
  }
}
```

Теперь давайте создадим экземпляр предыдущего класса и исполним методы для различных операций:

```
ContactMemcachedClient contactMemcachedClient = new
ContactMemcachedClient();

// Сохранить сведения о контактном лице
String valueTobeStored =
"jaydip,lakhatariya,jaydip@knowarth.com,9111111111,knowarth";
contactMemcachedClient.set(ContactMemcachedClient.NAMESPACE + "101",
valueTobeStored);

// Извлечь сведения о контактном лице
contactMemcachedClient.get(ContactMemcachedClient.NAMESPACE + "101");

// Удалить сведения о контактном лице
contactMemcachedClient.delete(ContactMemcachedClient.NAMESPACE + "101");
```

NoSQL API с PHP

С помощью PHP мы можем получать доступ к интерфейсу Memcached и управлять данными, так как NoSQL использует API, предоставляемые для MySQL 8. Для использования Memcached в приложениях на PHP необходимо установить расширение PECL. Ниже приведен пример программы на PHP, которая отображает список контактных лиц и имеет форму, которая будет хранить новую контактную информацию с помощью NoSQL API. Простой пример в программе PHP иллюстрирует основные операции, такие как сохранение новых контактных лиц и удаление контактных лиц.

Давайте создадим файл `init.php` со следующим ниже содержимым для подключения к экземпляру Memcached:

```
<?php
  $memcache = new Memcache;
  $memcache->addServer('10.12.4.15',11211);
?>
```

Теперь создадим еще один файл `addContact.php` для добавления данных в базу данных:

```
<?php require "init.php"; ?>
<?php
  if (isset($_POST['submit'])) {
    $key = $_POST['contactId'];
    $value = $_POST['firstName'];
    $value .= ",". $_POST['lastName'];
    $value .= ",". $_POST['emailAddress'];
    $value .= ",". $_POST['mobileNumber'];
    $value .= ",". $_POST['companyName'];
```

```
    $memcache->set($key,$value);
  }
?>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Add Contact</title>
  </head>
  <body>
    <form method="post">
      <p><b>ИД контакта</b>: <input type="text" size="20"
                                    name="contactId"></p>
      <p><b>Имя</b>: <input type="text" size="20"
                            name="firstName"></p>
      <p><b>Фамилия</b>: <input type="text" size="20"
                                name="lastName"></p>
      <p><b>Электронный адрес</b>: <input type="text" size="20"
                                          name="emailAddress"></p>
      <p><b>Мобильный номер</b>: <input type="text" size="20"
                                        name="mobileNumber"></p>
      <p><b>Название компании</b>: <input type="text" size="20"
                                          name="companyName"></p>
      <input type="submit" name="submit" value="submit">
    </form>
  </body>
</html>
```

Мы можем создать `deleteContact.php`, который будет удалять данные:

```
<?php
  if (isset($_POST['contactId'])) {
    $contact = $memcache->get($_POST['contactId']);
    if(isset($_POST['contactId'])) {
      $memcache->delete($_POST['contactId']);
    }
  }
?>
```

NoSQL API с Python

Python является полнофункциональным языком программирования высокого уровня и широко используется в анализе данных, искусственном интеллекте, научных вычислениях и приложениях для работы с большими данными. MySQL 8 имеет поддержку Memcached API для языка Python, которая может выполнять различные варианты использования, связанные с большими данными. Чтобы использовать Memcached API с языком Python, нам нужно установить модуль Python `Memcached`. Давайте испытаем некоторые части из Memcached API с языком Python для управления контактами.

Извлечь контактную информацию в командной строке и сохранить ее посредством Memcached API с помощью следующего ниже фрагмента кода:

```
import sys
import memcache
memcached = memcache.Client(['127.0.0.1:11211'], debug=1);
```

```
# Вставить сведения о контактном лице
def addContact(contactId, firstName, lastName, emailAddress, mobileNumber,
               companyName):
    valueTobeStore = firstName + ","
    valueTobeStore = lastName + ","
    valueTobeStore = emailAddress + ","
    valueTobeStore = mobileNumber + ","
    valueTobeStore = companyName
    memcached.set(contactId,valueTobeStore)
    print("Контактная информация сохранена успешно")

# Принять контактную информацию из командной строки
contactId= sys.argv[0]
firstName= sys.argv[1]
lastName= sys.argv[2]
emailAddress = sys.argv[3]
mobileNumber = sys.argv[4]
companyName = sys.argv[5]

# Вызвать функцию addContact, чтобы сохранить контактную информацию
addContact(contactId, firstName, lastName, emailAddress, mobileNumber, companyName)
```

Получить контактную информацию по идентификатору, переданному через аргументы командной строки, как указано в следующем ниже примере:

```
import sys
import memcache
memcached = memcache.Client(['127.0.0.1:11211'], debug=1);

# Извлечь контактную информацию по идентификатору
def getContactById(contactId):
    contact = memcached.get(contactId)
    print(contact)

# Принять идентификатор контактного лица из командной строки
contactId= sys.argv[0]
getContactById(contactId)
```

Удалить контакты по их идентификаторам, которые были переданы через аргумент командной строки, как показано в следующем ниже примере:

```
import sys
import memcache
memcached = memcache.Client(['127.0.0.1:11211'], debug=1);

def deleteContact(contactId)
    memcached.delete(contactId)
    print("Контактная информация была успешно удалена")

# Принять контактную информацию из командной строки
contactId= sys.argv[0]
deleteContact(contactId)
```

NoSQL API с Perl

Для доступа к интерфейсу Memcached приложения на основе Perl должны иметь модуль `Cache::Memcached`. Его можно установить с помощью следующей ниже команды:

```
perl -MCPAN -e 'install Cache::Memcached'
```

Как только этот модуль будет успешно установлен, мы можем использовать протокол Memcached через родной интерфейс.

Давайте посмотрим на пример, который принимает контактную информацию из командной строки и сохраняет информацию в Memcached, после чего мы проверим сохраненную информацию, извлекая ее по ключу:

```
#!/usr/bin/perl
use Cache::Memcached;
use Data::Dumper;

# Сконфигурировать сервер memcached
my $memcache = new Cache::Memcached {
'servers' => [ 'localhost:11211']
};

my $contactId=$ARGV[0];
my $firstName=$ARGV[1];
my $lastName=$ARGV[2];
my $emailAddress=$ARGV[3];
my $mobileNumber=$ARGV[4];
my $companyName=$ARGV[5];

# Сохранить контактную информацию
$contactData = $firstName . ",". $lastName . "," . $emailAddress . "," .
$mobileNumber . "," . $companyName;
$memcache->set($contactId,$contactData))

# Извлечь контактную информацию для проверки, сохранена она или нет
my $contact = $cache->get($contactId);
if (defined($contact))
{
  print "$contact\n";
}
else
{
  print STDERR "Контактная информация не сохранена";
}
```

NDB API Cluster

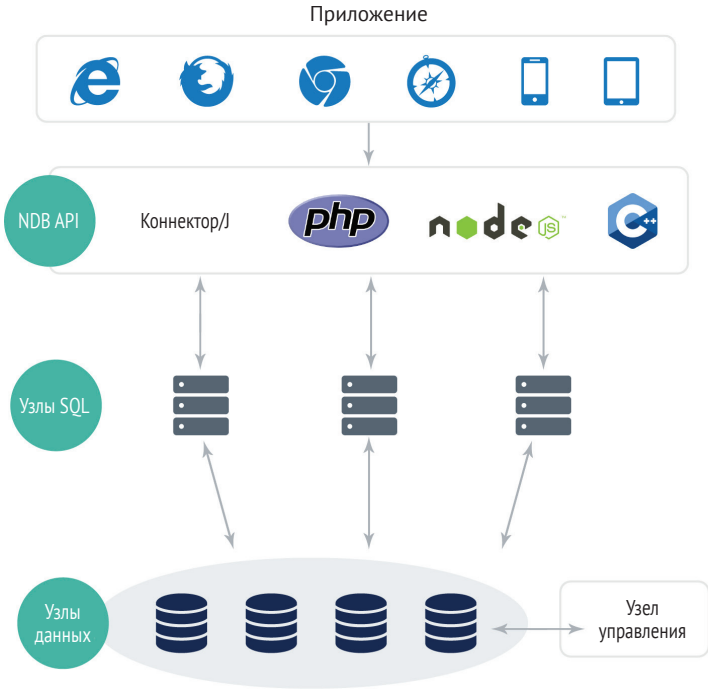
Кластеризация – это механизм распределения нагрузки между несколькими серверами. При работе с большими данными предприятиям всегда нужно думать о высокой доступности, производительности и масштабируемости. Архитектура с одним узлом может не удовлетворять этого требования в отношении доступности, масштабируемости и производительности. Как показано ниже, есть некоторые вопросы, которые могут возникнуть с архитектурой одного узла:

- Что делать, если один узел отключается?
- Обеспечит ли один узел хорошую производительность, так как все запросы должны обслуживаться только одним сервером?
- Как мы можем сделать его масштабируемым, если запросная нагрузка увеличится?

NDB Cluster, который обозначается как **сетевая база данных**, активизирует кластеризацию базы данных в памяти с системой «без разделения» (shared-nothing). Это позволяет легко обрабатывать сценарии высокой доступности и масштабируемости, а также повышать производительность при распределении нагрузки между несколькими серверами. Система без разделения означает, что многочисленные узлы связаны между собой с целью распределенных вычислений с их собственной памятью и собственным хранилищем.

Система MySQL Cluster построена на подсистеме хранения данных NDB или NDB Cluster, которые обеспечивают низкую задержку, высокую пропускную способность, масштабируемость и высокую доступность. Она принимает горизонтальное масштабирование и автоматическое распределение для обслуживания операций чтения/записи большой нагрузки через различные запросы NoSQL. NDB Cluster – это набор различных узлов, где каждая задача выполняется на собственном процессоре.

Узел – это просто часть кластера, которая может быть классифицирована как узел управления, узел данных и узел SQL:



Как описано в схеме архитектуры NDB Cluster, узлы данных отвечают за хранение данных. Эти данные могут быть доступны посредством различных NoSQL API на различных языках программирования, таких как NodeJS, Java, C++ и HTTP/REST, так называемых NDP API. Давайте закатаем рукава и займемся исследованием NDP API с различными языками.

NDB API для NodeJS

NDB Cluster 7.3 ввел API для приложений, написанных на JavaScript с использованием NodeJS. Коннектор MySQL для JavaScript включает в себя адаптер, кото-

рый может непосредственно обращаться к подсистеме хранения данных NDB. Этот интерфейс является асинхронным со встроенной моделью событий NodeJS. NodeJS – это платформа, которая имеет действительно хорошую производительность, поскольку она была построена на движке JavaScript Google Chrome V8.

Драйвер NDB Cluster на JavaScript для NodeJS упрощает прямой доступ к данным из JavaScript. Следовательно, удалена лишняя задержка за счет синтаксического анализа объектов по операциям SQL, и он может непосредственно читать/писать данные из узлов данных NDB Cluster. Для запуска приложения NodeJS с NDP API необходимо установить адаптер `mysql-js` и драйвер `node-mysql`.

Построим простое приложение NodeJS для выполнения некоторых операций по управлению контактными лицами с помощью следующего ниже примера.

Прежде всего нам нужно импортировать модуль `mysql-js`, создать класс как `Contact` с его атрибутами и увязать его с таблицей, как показано в следующем ниже фрагменте кода:

```
var nosql = require('mysql-js');

var Contact = function(contactId, firstName, lastName, emailAddress,
mobileNumber, companyName) {
      if (contactId) this.contactId = contactId;
      if (firstName) this.firstName = firstName;
  if (lastName) this.lastName = lastName;
  if (emailAddress) this.emailAddress = emailAddress;
  if (mobileNumber) this.mobileNumber = mobileNumber;
  if (companyName) this.companyName = companyName;
};

// Увязать класс с таблицей
var annotations= new nosql.TableMapping('Contacts').applyToClass(Contact);
```

Создать функцию обратного вызова, которая будет вызвана после того, как будет попытка создания подключения к MySQL Cluster:

```
var onOpenSession = function(err, session) {
  if (err) {
    console.log('Ошибка при открытии сеанса');
        console.log(err);
        process.exit(0);
    } else {
        console.log('Подключение успешно установлено');
        // Сохранить контактную информацию
        var contact = new Contact(123, 'Jaydip', 'Lakhatariya',
'jaydip.lakh@gmail.com', '9111111111', 'Knowarth Tech');
        session.persist(contact, onInsert, contact, session);
    }
};
```

Как показано в предыдущем фрагменте кода, если успешное подключение устанавливается, то будет вставлена новая контактная информация и вызвана следующая ниже функция обратного вызова:

```
var onInsert = function(err, object, session) {
  if (err) {
```

```
      console.log("Ошибка при вставке новой контактной информации")
      console.log(err);
  } else {
      console.log('Контактная информация успешно вставлена');

      // Найти контактную информацию в базе данных
      session.find(Contact, 123, onFindContact);
  }
};
```

Создать функцию обратного вызова, которая будет вызываться всякий раз, когда выполняется операция поиска `Find`:

```
var onFindContact = function(err, contact) {
    if (err) {
        console.log("Ошибка при извлечении контакта")
        console.log(err);
    } else {
        console.log('Найденная контактная информация: ' +
JSON.stringify(contact));
        // Удалить соответствующую контактную информацию
        session.remove(contact, onDeleteContact, contact.contactId);
    }
};
```

Если предыдущий метод достигает успеха в получении контактной информации, то он попытается удалить этот же контакт, и при этом будет выполнена следующая ниже функция обратного вызова:

```
var onDeleteContact = function(err, object) {
  if (err) {
    console.log("Ошибка при удалении контактной информации");
    console.log(err);
  } else {
    console.log('Успешно удален контакт ИД: ' + object.contactId);
  }
  process.exit(0);
};
```

В конце нам нужно инициализировать и открыть подключение для `ndb` с помощью следующего ниже фрагмента кода:

```
// Инициализировать свойства базы данных
var dbProperties = nosql.ConnectionProperties('ndb');

// Подключиться к базе данных
nosql.openSession(dbProperties, Contact, onOpenSession);
```

JavaScript может легко работать с объектами, поэтому обеспечивает простой способ увязывать данные таблицы непосредственно с объектом. Мы можем получить доступ к информации таблицы в виде объекта. В нашем примере у нас есть класс `Contact`, определенный с различными атрибутами, которые были увязаны с таблицей MySQL `Contacts`, используя класс `TableMapping`.

Мы должны определить `ndb` как свойства подключения, чтобы непосредственно обратиться к узлам данных, а не идти на сервер MySQL. Метод `openSession` создает

мост между API и узлами данных. С помощью метода `openSession` мы также можем указать функцию обратного вызова, поскольку JavaScript является событийно-управляемым языком программирования.

Давайте выполним созданную программу и посмотрим на результат:

```
[root@ip-172-31-22-59 ~]$ node my-test.js
Подлючение успешно установлено
Контактная информация успешно вставлена:
Найдена контактная информация:
{"contactId":"123","firstName":"Jaydip","lastName":"Lakhatariya","emailAddr
ess":"jaydip.lakh@gmail.com","phoneNumber":"9724371151","companyName":"KNOW
ARTH"}
Успешно удален контакт ИД: 123
>
```

NDB API для Java

Основанные на Java корпоративные приложения могут легко получать доступ к MySQL Cluster, используя NoSQL API с помощью интерфейса ClusterJ. Этот интерфейс предназначен для обеспечения высокой производительности при сохранении и извлечении информации непосредственно из базы данных MySQL Cluster. Его очень легко применять разработчикам, использующим родной интерфейс Java и плагин JPA, потому что он автоматически предоставляет преобразование таблично-ориентированных представлений данных, хранящихся в MySQL Cluster, в используемые приложением объекты Java.

Чтобы выполнить преобразование между таблицами и объектами Java, необходимо аннотировать интерфейсы, представляющие объекты Java, где каждый интерфейс увязан с таблицей, в то время как каждое свойство этого интерфейса увязано со столбцами таблицы, так же как в интерфейсе JPA/Hibernate.

Коннектор `clusterj` поставляется вместе с MySQL Cluster. После установки MySQL Cluster необходимые библиотеки `jar` доступны в папке MySQL `/share/java`. Перейдите к этой папке, скопируйте файлы `clusterj.jar` и `clusterj-api.jar` и поместите их в свое приложение. Попробуем интегрировать `clusterj` для контактной информации.

Прежде всего мы должны предоставить определенные свойства, которые описывают, как библиотека `clusterj` должна подключаться к MySQL Cluster, такие как URL-адрес подключения, номер порта, какие базы данных использовать, детали аутентификации и другие атрибуты типа тайм-аута и максимального количества транзакций.

Ниже приведен пример файла с именем `clusterj.properties`, имеющийся в исходном коде приложения:

```
com.mysql.clusterj.connectstring=localhost:1186
com.mysql.clusterj.database=demodb
com.mysql.clusterj.connect.retries=4
com.mysql.clusterj.connect.delay=5
com.mysql.clusterj.connect.timeout.before=30
com.mysql.clusterj.connect.timeout.after=20
com.mysql.clusterj.max.transactions=1024
```

Создайте модельный интерфейс для увязки с таблицей в соответствии со следующим ниже программным кодом. Имя таблицы может быть автоматически увяза-

но с модельным интерфейсом с помощью аннотации `@PersistenceCapable`. Кроме того, имя столбца можно увязать с аннотацией `@Column` библиотеки clusterJ.

```
import com.mysql.clusterj.annotation.Column;
import com.mysql.clusterj.annotation.PersistenceCapable;
import com.mysql.clusterj.annotation.PrimaryKey;

@PersistenceCapable(table="Contacts")
public interface Contact {
  @PrimaryKey
  public int getId();
  public void setId(int id);

  @Column(name="firstName")
  public String getFirstName();
  public String setFirstName(String firstName);
  @Column(name="lastName")
  public String getLastName();
  public String setLastName(String lastName);
  @Column(name="emailAddress")
  public String getEmailAddress();
  public String setEmailAddress(String emailAddress);
  @Column(name="mobileNumber")
  public String getMobileNumber();
  public String setMobileNumber(String mobileNumber);
  @Column(name="companyName")
  public String getCompanyName();
  public String setCompanyName(String companyName);
}
```

Теперь все готово, для того чтобы взглянуть на основные операции с помощью следующего ниже примера:

```
import com.mysql.clusterj.ClusterJHelper;
import com.mysql.clusterj.SessionFactory;
import com.mysql.clusterj.Session;
import com.mysql.clusterj.Query;
import com.mysql.clusterj.query.QueryBuilder;
import com.mysql.clusterj.query.QueryDomainType;
import java.io.File;
import java.io.InputStream;
import java.io.FileInputStream;
import java.io.*;
import java.util.Properties;
import java.util.List;

public class clusterJ {

  private static Session session;
  public static void main (String[] args) {
    // Загрузить свойства из файла clusterj.properties
    File propsFile = null;
    InputStream inStream = null;
    Properties props = null;
    props.load(inStream);
```

```
    try {
      propsFile = new File("clusterj.properties");
      inStream = new FileInputStream(propsFile);
      props = new Properties();
      props.load(inStream);
    } catch(Exception e) {
      System.err.println(e);
      return;
    }
    // Подключиться к базе данных и создать сеанс
    SessionFactory factory = ClusterJHelper.getSessionFactory(props);
    session = factory.getSession();
    if(session != null) {
      System.out.println("Подключение успешно установлено");
    }
    saveContact(123, "Jaydip", "Lakhatariya", "jaydip.lakh@gmail.com",
"9724371151", "Knowarth");
    findByContactId(123);
    deleteContactById(123);
  }
}
```

Ниже приведен метод сохранения контактной информации:

```
private static void saveContact(int contactId, String firstName, String
lastName, String emailAddress, String mobileNumber, String companyName) {
  // Создать и инициализировать контактную информацию
  Contact newContact = session.newInstance(Contact.class);
  newContact.setId(contactId);
  newContact.setFirstName(firstName);
  newContact.setLastName(lastName);
  newContact.setEmailAddress(emailAddress);
  newContact.setMobileNumber(mobileNumber);
  newContact.setCompanyName(companyName);
  // Сохранить контактную информацию в базе данных
  session.persist(newContact);
  System.out.println("Контактная информация успешно сохранена");
}
```

Этот метод отвечает за поиск контактной информации на основе ее идентификатора:

```
private static void findByContactId(int contactId) {
  Contact contact = session.find(Contact.class, contactId);
  if(contact != null) {
    System.out.println("Найден контакт с именем: " +
      contact.getFirstName() + " " + contact.getLastName());
  } else {
    System.out.println("Не найден контакт с ИД " + contactId);
  }
}
```

Этот метод будет отвечать за удаление контактной информации по ее идентификатору:

```
private static void deleteContactById(int contactId) {
  session.deletePersistent(Contact.class, contactId);
  System.out.prinln("Удалена контактная информация с ИД: " + contactId);
}
```

Давайте исполним приведенную выше программу и посмотрим на результат:

```
Подключение успешно установлено
Контактная информация успешно сохранена
Найден контакт с именем: Jaydip Lakhatariya
Удалена контактная информация с ИД: 123
```

Ниже приведено краткое описание классов clusterJ, используемых в предыдущей программе:

- **SessionFactory**: один экземпляр в расчете на MySQL Cluster и используется для хранения сеансов. С этим классом связаны все свойства конфигурации кластера.
- **ClusterJHelper**: предоставляет вспомогательные методы для моста между API и реализацией.
- **Session**: один экземпляр в расчете на пользователя и представляет подключение кластера. Все операции можно выполнять только с помощью этого класса.

Библиотека clusterJ предоставляет хороший интерфейс для разработчиков, но она также имеет некоторые ограничения, которые вынуждают разработчиков использовать OpenJPA с плагином ClusterJPA. Вот ограничения библиотеки clusterJ:

- мы должны объявлять интерфейсы для объектов, а не для классов. Разработчик должен предоставлять сигнатуры методов get/set, а не свойства; мы также не можем размещать в интерфейсе дополнительные методы;
- мы не можем создавать связи между свойствами и объектами;
- поддерживается только одна-единственная табличная операция, где мы не можем выполнять многотабличное наследование;
- поскольку запросы ограничены одной таблицей, мы не можем выполнять соединения;
- она не поддерживает ленивую загрузку, поэтому будет загружать весь набор записей за один раз, включая крупные объекты.

NDB API с C++

NDB API для C++ может быть реализован с помощью библиотеки `NdbApi.hpp`, которая обеспечивает несколько классов для различных целей, как описано в следующих далее пунктах:

- `Ndb`: основной класс NDB API и представлен как ядро NDB Kernel;
- `Ndb_cluster_connection`: обрабатывает подключение к кластеру узлов данных;
- `NdbTransaction`: управляет транзакциями базы данных;
- `NdbOperation`: обрабатывает различные операции, такие как удаление кортежа, вставка кортежа, чтение кортежа, обновление кортежа и т. д.

Давайте рассмотрим пример использования NDB API на языке C++, где мы определим необходимые библиотеки, функции и макросы:

```
#include <NdbApi.hpp>
#include <stdio.h>
#include <iostream>
```

```
static void run_application(Ndb_cluster_connection &);
static void create_table(MYSQL &);
static void do_insert(Ndb &);
static void do_update(Ndb &);
static void do_delete(Ndb &);
static void do_read(Ndb &);

#define PRINT_ERROR(code,msg)
  std::cout << "Ошибка: " << code << ", Сообщение: " << message <<
std::endl

#define APIERROR(error) {
  PRINT_ERROR(error.code,error.message);
  exit(-1);
}
```

Давайте определим главный метод, который создаст подключение к MySQL Cluster и вызовет метод `run_application`:

```
int main(int argc, char** argv)
{
  // Initialize ndb
  ndb_init();

  // Connect MySQL Cluster
  {
    const char *connection_string = "localhost:1186";
    Ndb_cluster_connection cluster_connection(connection_string);

    // Подключить к серверу управления кластерами (ndb_mgmd)
    if (cluster_connection.connect(4, 5, 1))
    {
      std::cout << "Сервер управления еще не готов.\n";
      exit(-1);
    }

    // Подключается к узлам хранения (ndbd)
    if (cluster_connection.wait_until_ready(30,0) < 0)
    {

      std::cout << "Узлы данных еще не готовы.\n";
      exit(-1);
    }

    // Выполнить код приложения
    run_application(cluster_connection);
  }
  ndb_end(0);
  return 0;
}
```

Определить метод `run_application`, который инициализирует подключение к базе данных и выполнит различные операции:

```
static void run_application(Ndb_cluster_connection &cluster_connection)
{
  // Подключиться к базе данных через NDB API
```

```
  Ndb myNdb( &cluster_connection, "demodb" );
  if (myNdb.init()) APIERROR(myNdb.getNdbError());

  // Выполнить различные операции на базе данных
  do_insert(myNdb);
  do_update(myNdb);
  do_delete(myNdb);
  do_read(myNdb);
}
```

Метод do_insert добавляет новую контактную информацию:

```
static void do_insert(Ndb &myNdb)
{
  const NdbDictionary::Dictionary* myDict= myNdb.getDictionary();
  const NdbDictionary::Table *myTable= myDict->getTable("Contacts");
  if (myTable == NULL) APIERROR(myDict->getNdbError());

  NdbTransaction *myTransaction= myNdb.startTransaction();
  if (myTransaction == NULL) APIERROR(myNdb.getNdbError());
  NdbOperation *myOperation= myTransaction->getNdbOperation(myTable);
  if (myOperation == NULL) APIERROR(myTransaction->getNdbError());
  myOperation->insertTuple();
  myOperation->equal("id", 123);
  myOperation->setValue("firstName", "Jaydip");
  myOperation->setValue("lastName", "Lakhatariya");
  myOperation->setValue("emailAddress", "jaydip.lakh@gmail.com");
  myOperation->setValue("mobileNumber", "9724371151");
  myOperation->setValue("companyName", "KNOWARTH");

  if (myTransaction->execute( NdbTransaction::Commit ) == -1)
    APIERROR(myTransaction->getNdbError());
  myNdb.closeTransaction(myTransaction);
}
```

Метод do_update обновляет существующую запись с новой контактной информацией:

```
static void do_update(Ndb &myNdb)
{
  const NdbDictionary::Dictionary* myDict= myNdb.getDictionary();
  const NdbDictionary::Table *myTable= myDict->getTable("Contacts");
  if (myTable == NULL) APIERROR(myDict->getNdbError());

  NdbTransaction *myTransaction= myNdb.startTransaction();
  if (myTransaction == NULL) APIERROR(myNdb.getNdbError());

  NdbOperation *myOperation= myTransaction->getNdbOperation(myTable);
  if (myOperation == NULL) APIERROR(myTransaction->getNdbError());

  myOperation->updateTuple();
  myOperation->equal( "id", 123);
  myOperation->setValue("firstName", "Keyur");
  myOperation->setValue("lastName", "Lakhatariya");
  myOperation->setValue("emailAddress", "Keyur.lakh@gmail.com");
  myOperation->setValue("mobileNumber", "9998887771");
```

```
  myOperation->setValue("companyName", "KNOWARTH");

  if( myTransaction->execute( NdbTransaction::Commit ) == -1 )
    APIERROR(myTransaction->getNdbError());
  myNdb.closeTransaction(myTransaction);
}
```

Метод do_delete удаляет контактную информацию с идентификатором 123:

```
static void do_delete(Ndb &myNdb)
{
  const NdbDictionary::Dictionary* myDict= myNdb.getDictionary();
  const NdbDictionary::Table *myTable= myDict->getTable("Contacts");
  if (myTable == NULL)
    APIERROR(myDict->getNdbError());

  NdbTransaction *myTransaction= myNdb.startTransaction();
  if (myTransaction == NULL) APIERROR(myNdb.getNdbError());
  NdbOperation *myOperation= myTransaction->getNdbOperation(myTable);
  if (myOperation == NULL) APIERROR(myTransaction->getNdbError());
  myOperation->deleteTuple();
  myOperation->equal( "id", 123);
  if (myTransaction->execute(NdbTransaction::Commit) == -1)
    APIERROR(myTransaction->getNdbError());
  myNdb.closeTransaction(myTransaction);
}
```

Метод read извлекает запись с идентификатором контактного лица 123 и отображает его адрес электронной почты и номер мобильного телефона:

```
static void do_read(Ndb &myNdb)
{
  const NdbDictionary::Dictionary* myDict= myNdb.getDictionary();
  const NdbDictionary::Table *myTable= myDict->getTable("Contacts");
  if (myTable == NULL)
    APIERROR(myDict->getNdbError());

    NdbTransaction *myTransaction= myNdb.startTransaction();
    if (myTransaction == NULL) APIERROR(myNdb.getNdbError());
    NdbOperation *myOperation= myTransaction->getNdbOperation(myTable);
    if (myOperation == NULL) APIERROR(myTransaction->getNdbError());
    myOperation->readTuple(NdbOperation::LM_Read);
    myOperation->equal("id", 123);
    NdbRecAttr *emailRecAttr= myOperation->getValue("emailAddress", NULL);
  NdbRecAttr *mobileRecAttr= myOperation->getValue("mobileNumber", NULL);
    if (emailRecAttr == NULL) APIERROR(myTransaction->getNdbError());
    if (mobileRecAttr == NULL) APIERROR(myTransaction->getNdbError());

    if(myTransaction->execute( NdbTransaction::Commit ) == -1)
      APIERROR(myTransaction->getNdbError());

    printf("Имя: \n", emailRecAttr->aRef());
    printf("Мобильный номер: \n", mobileRecAttr->aRef());
    myNdb.closeTransaction(myTransaction);
}
```

Таким образом, предприятия, работающие на разных языках программирования, могут интегрировать API со своими приложениями, чтобы работать как NoSQL. Он может быть легко интегрирован с односерверным узлом или кластерной средой с помощью интерфейса Memcached или NDB API. Разработчики могут выбирать интерфейсы API на основе предпочитаемого языка для простой обработки больших данных. Он действительно обеспечивает лучшую производительность, поскольку обходит слой SQL и использует данные непосредственно. С помощью этих API мы можем использовать MySQL в качестве базы данных NoSQL.

Резюме

Прочитав эту главу, мы теперь можем быть уверены в том, каким образом можно использовать MySQL в качестве базы данных NoSQL с нашим приложением, которое построено на разных языках программирования. В начале этой главы мы обсудили базу данных NoSQL и ее основные преимущества для использования в наших приложениях. Позже мы увидели различия между SQL и NoSQL, а затем приступили к изучению различных API на основе элементарных примеров. Мы разобрались с Memcached API на простых примерах с использованием различных языков программирования, таких как Java, PHP, Python и Perl, которые предоставляют доступ к MySQL в качестве базы данных NoSQL. Затем мы познакомились с различными NoSQL API, используемыми для управления базой данных для кластерной среды. Мы обсудили NDB API и то, как его можно использовать с различными языками программирования, такими как NodeJS, Java и C++. Эти различные API с поддержкой NoSQL дают возможность разработчикам и предприятиям с легкостью работать с большими данными в MySQL.

В следующей главе мы разберемся в том, как использовать Hadoop, чтобы помочь MySQL в обработке больших данных, и как применять Apache Sqoop для передачи данных между Hadoop и MySQL. Мы увидим это на практическом примере из реальной жизни.

Глава 9

Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop

В предыдущей главе мы познакомились с различными методами использования NoSQL API для MySQL. Мы рассмотрели примеры PHP, JSON и Java для подключения и использования данных с применением NoSQL API. В этой главе мы познакомимся с тем, как использовать Apache Sqoop и Hadoop в жизненном цикле больших данных для обработки неструктурированных данных в структурированные, которыми можно легко манипулировать с помощью реляционных систем управления базами данных, таких как MySQL.

Ниже приведены темы, которые мы собираемся затронуть в этой главе:

- практический пример анализа журналов операций;
- обзор Apache Sqoop;
- интеграция Apache Sqoop с MySQL и Hadoop;
- импорт неструктурированных данных в Hadoop HDFS из MySQL;
- загрузка структурированных данных в MySQL с помощью Apache Sqoop.

В главе 1 «*Введение в большие данные и MySQL 8*» мы узнали, как MySQL вписывается в жизненный цикл больших данных. Предположим, мы создаем систему анализа настроений людей, опираясь на сделанные пользователями твиты и сообщения Facebook. Как мы уже знаем, каждую секунду пользователи отправляют миллионы твитов и сообщений Facebook, поэтому наша первая задача – захватить всю эту информацию и сохранить ее где-то в базе данных. Затем следующим шагом будет правильно расклассифицировать неструктурированную информацию на основе теговых ключевых слов, используемых пользователями в своих сообщениях. Третий шаг заключается в том, чтобы сохранить эти данные обратно в структурированную базу данных, где мы можем создать соответствующие связи и ссылочную целостность данных.

На шаге 1 для передачи неструктурированных данных из MySQL в Hadoop для быстрой обработки данных используется Apache Sqoop.

Практический пример анализа журналов операций

Еще одним примером анализа крупных массивов данных для коммерческого использования является анализ журналов операций. Рассмотрим приложение электронной коммерции, где тысячи пользователей посещают сайты ежедневно. Не все 100 процентов пользователей, посещающих приложение, собираются приобрести товар. Люди, посещающие приложения электронной коммерции, как правило, большую часть времени делают следующие действия:

- приобретают товар;
- просматривают товары, приобретенные их друзьями и/или родственниками;
- читают отзывы других людей о товаре, который они надеются приобрести;
- ищут товары с предложениями по наиболее выгодной цене.

Поэтому человек, который еще не приобрел товары, считается потенциальным покупателем. Очень важно привлечь таких потенциальных покупателей и конвертировать их визит в реальную продажу. Таким образом, в этом случае, если приложение может определить, какие товары просматриваются потенциальными покупателями и на какой конкретный товар или категорию они потратили максимальное время, на основе их посещений страницы приложение сможет выявить аналогичные товары по наиболее выгодной цене или популярные и наиболее продаваемые товары в аналогичных категориях. Пользователям будет очень интересно, когда они увидят варианты товаров, которые они ищут, и это обеспечит плавный опыт пользования приложением.

Еще один случай анализа журнала операций пользователя для приложения электронной коммерции – это сокращение количества брошенных корзин в приложении. Брошенная корзина означает, что пользователь добавил несколько элементов в свою корзину, но не прошел этап подтверждения заказа и на определенном этапе оставил корзину бесхозно. Важно определить, на каком этапе совершения покупок пользователи сталкиваются с трудностями при подтверждении заказа. Такой анализ журнала операций может помочь магазину электронной коммерции определить страницы, где большинство пользователей покинуло корзину, или даже определить, какие страницы занимают большую часть времени. В среднем на приложения электронной коммерции пользователь тратит от 5 до 10 минут времени, поэтому очень важно, чтобы в течение этого периода они попадали на нужные страницы.

На основе анализа журнала операций ниже приведены показатели, которые мы можем определить для приложения:

- среднее время, которое посетитель тратит на приложение;
- среднее число страниц, посещенных посетителями за сеанс;
- среднее число ежедневных посетителей;
- день недели или месяц года, когда посещаемость приложения наивысшая;
- поток просмотровых действий пользователя для определения показателя брошенных страниц на разных этапах просмотра.

Использование MySQL 8 и Hadoop для анализа журналов операций

Так как мы узнали ранее, что анализ журналов операций может заставить журнал операций пользователя приносить доход, то очень важно выяснить, как такая

информация может храниться в базе данных. Информация журнала операций генерируется, когда пользователь просматривает приложение, поэтому мы получаем информацию журнала в сыром виде. Для идентификации нужной информации требуются некоторые дополнительные вычисления в журнале операций. Ежедневно в приложении мы получаем миллионы нажатий, поэтому делать такие аналитические вычисления в реляционной базе данных, такой как MySQL, может оказаться сложным, поскольку это может приводить к дополнительной на нее нагрузке. Самый лучший подход здесь состоял бы в том, чтобы дать MySQL 8 обслуживать клиентскую часть приложения и собирать всю информацию журнала операций в своем сыром виде, используя другие доступные технологии, которые специально спроектированы для обработки большого набора данных. Такие технологии больших данных, как Hadoop, могут вычислять крупный объем данных за считанные секунды.

Поэтому что мы можем здесь сделать, так это сохранить все сырые данные, которые мы получили из журнала операций в MySQL 8. Эта информация журнала будет неструктурированной и не будет содержать никаких полезных сведений. Затем мы можем перенести этот неструктурированный журнал операций в Apache Hadoop. Apache Hadoop – это платформа больших данных с открытым исходным кодом, реструктуризирующая данные в формат, который может быть представлен пользователям. Как только данные преобразуются в презентабельный формат, используя алгоритмы Hadoop, такие как MapReduce, мы можем передать хорошо структурированные данные обратно в MySQL 8. Затем на основе этих структурированных данных в MySQL 8 могут быть подготовлены разные отчеты для извлечения различной информации. Для передачи большого объема данных из MySQL 8 в Hadoop мы можем использовать Apache Sqoop. Apache Sqoop – это опять-таки технология с открытым исходным кодом для обмена данными между реляционными и нереляционными базами данных. Мы узнаем больше о Apache Sqoop, Hadoop и как обмениваться данными между Hadoop и MySQL 8 с помощью Sqoop далее в этой главе.

Обзор Apache Sqoop

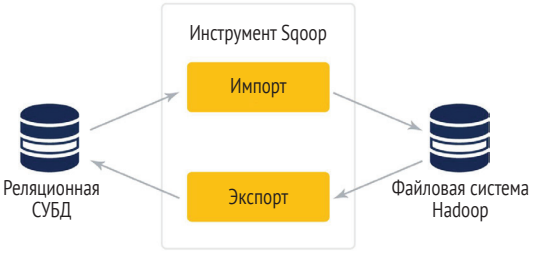
Реляционные СУБД, такие как MySQL, наиболее популярны во время работы с подавляющей частью деловых приложений. Поскольку СУБД MySQL имеет открытый исходный код, ее выбирают по умолчанию для многих приложений. Как уже отмечалось в главе 1 «*Введение в большие данные и MySQL 8*», приложения, построенные с использованием MySQL в качестве СУБД, могут принадлежать к разным предметным областям, таким как электронная коммерция, электронное обучение, медицинские и здравоохранительные информационные сайты и социально-медийные приложения. Каждая из этих предметных областей обладает мощностью генерировать большое количество данных, которые должны храниться в MySQL. Некоторые из этих данных структурированы, как, например, регистрационная информация пользователя или информация корзины пользователя, которая хорошо поддерживается MySQL. С другой стороны, некоторые данные не структурированы, например информация о последней странице пользователя, лайки, акции или твиты. Информация о доступе этого пользователя может быть очень хорошо использована с целью улучшения пользовательского опыта при исполь-

зовании приложения. Это делается путем анализа поведения пользователя или создания рекомендательного механизма на основе предсказательного анализа сгенерированных данных.

Как мы уже говорили ранее, аналитические данные, генерируемые приложением, должны храниться в MySQL в сыром или неструктурированном формате. Эти неструктурированные данные, хранящиеся в MySQL, должны быть переданы в технологии больших данных, таких как Hadoop, для быстрого их анализа и реструктуризации, используя различные алгоритмы, доступные в технологиях больших данных. Передача этих больших данных может быть выполнена либо путем создания пользовательских программ, либо с помощью других готовых к использованию инструментов. Apache Sqoop является одним из таких инструментов, которые могут быть использованы для передачи данных из MySQL.

Apache Sqoop используется для эффективной передачи больших объемов данных между Apache Hadoop и реляционными системами управления базами данных, такими как MySQL, Oracle и Postgres. Sqoop может импортировать или экспортировать данные параллельно, чтобы ускорить передачу данных между исходной и конечной базами данных. Результаты процесса импорта Sqoop могут быть сохранены либо в текстовые файлы с разделителями, либо в двоичные файлы Avro или SequenceFile. SequenceFile содержат данные в сериализованном формате.

Ниже приведен простой рисунок, который кратко объясняет связь между MySQL, Sqoop и Hadoop:



Sqoop предоставляет пользователю интерфейс командной строки для выполнения различных действий по импорту или экспорту данных между Hadoop и MySQL. Sqoop предоставляет несколько функциональных возможностей, таких как:

- параллельная передача данных;
- инкрементный импорт;
- импорт на основе запросов SQL `SELECT`;
- поддержка различных строк подключения для различных реляционных баз данных.

Процесс импорта Sqoop выполняется параллельно, из-за чего результатом процесса импорта Sqoop является несколько сгенерированных файлов. Эти сгенерированные файлы могут использоваться распределенной файловой системой Hadoop для управления данными, скопированными из MySQL. Процесс импорта также создает класс `java`. Этот класс `java` имеет возможность сериализовать или десериализовать данные и читать текст с разделителями.

После обработки импортированных данных в распределенной файловой системе Hadoop их можно импортировать обратно в MySQL. Как и процесс импорта,

процесс экспорта тоже обрабатывает несколько файлов параллельно, вставляя данные обратно в MySQL. Данные, вставленные в MySQL, теперь хорошо структурированы и могут использоваться соответствующими клиентскими приложениями для создания различных аналитических отчетов.

Интеграция Apache Sqoop с MySQL и Hadoop

Apache Sqoop может работать только в том случае, если на сервере установлен Hadoop. Для работы Apache Sqoop требуется операционная система на базе Linux. Чтобы Hadoop и Sqoop работали на сервере Linux, необходимо установить Java. После установки Sqoop на сервере нам нужно скачать коннектор MySQL для Sqoop, который позволит драйверу JDBC подключаться к базе данных MySQL для передачи данных с Hadoop.

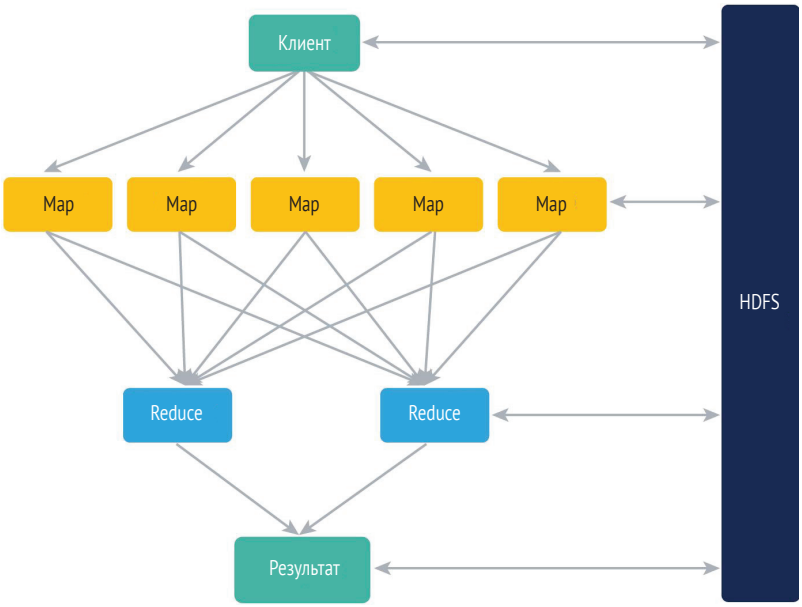
Hadoop

Hadoop – это платформа больших данных с открытым исходным кодом для быстрой обработки и анализа наборов данных крупного объема с помощью кластера среды. Благодаря среде Hadoop с многочисленными ведомыми узлами легко избежать аварийного прекращения работы системы или потери данных, если один или несколько узлов откажут. Hadoop в основном работает с несколькими модулями, такими как **YARN** (Yet Another Resource Negotiator, еще один ресурсный посредник), **распределенная файловая система Hadoop** (HDFS) и алгоритм **MapReduce**. Для параллельной обработки данных используется алгоритм Hadoop MapReduce. MapReduce применяется для преобразования неструктурированных данных в структурированный формат с использованием различных алгоритмов MapReduce.

MapReduce

Алгоритм Hadoop **MapReduce** используется для параллельной обработки крупного объема данных в кластеризованной среде, где данные размещаются в нескольких узлах. Это обеспечивает большую масштабируемость кластера Hadoop. Алгоритм MapReduce выполняет две отдельные задачи: Map и Reduce. В задаче Map алгоритм принимает неструктурированные данные в качестве входных данных, обрабатывает их и преобразовывает в пары ключ-значение.

На следующем ниже рисунке показана задача MapReduce. Он показывает, как задача от клиента делится на несколько преобразований, которые выполняют вычислительную работу на входе, а затем алгоритм сокращения уменьшает количество процессов до минимума, чтобы распределить результат на выходе.



Распределенная файловая система Hadoop

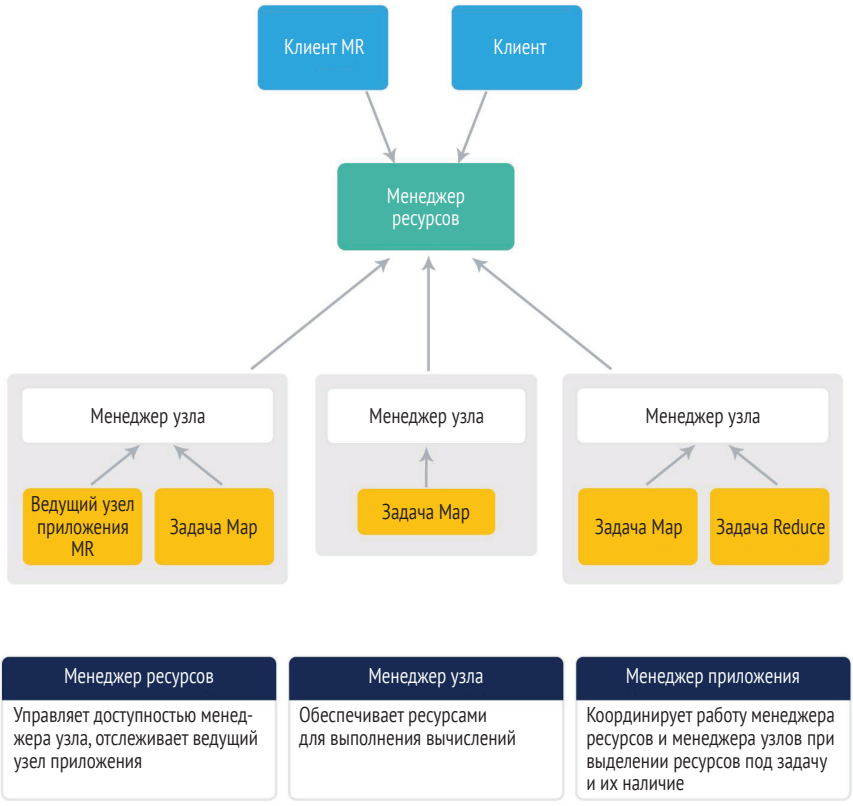
Распределенная файловая система Hadoop (HDFS) – это система хранения данных, используемая Hadoop. HDFS состоит из кластеров узлов для хранения миллиардов записей в системе. Информация о данных разбивается на несколько блоков, и эти блоки обрабатываются параллельно, ускоряя передачу данных. Кроме того, каждый блок данных копируется на многочисленные узлы, чтобы в случае аварийного прекращения работы одного узла данные могли считываться с других узлов до тех пор, пока не будет восстановлен аварийный узел. HDFS использует архитектуру «ведущий-ведомый».

Ведущий узел называется **узлом имен**, а ведомые узлы называются узлами данных. Узел имен не состоит из каких-либо данных, но содержит информацию о дереве файлов данных, и его блочная структура копируется в HDFS. Фактические данные хранятся в многочисленных ведомых узлах или узлах данных.

YARN

YARN, или «Еще один ресурсный посредник», используется для управления кластерными ресурсами. Начиная с Apache Hadoop 2, службу YARN можно разделить на две категории: менеджер ресурсов и менеджер узлов. Менеджер ресурсов управляет ресурсами, которые должны быть назначены различным приложениям, а также отслеживает менеджеров рабочих узлов. Менеджер узлов предоставляет необходимые ресурсы в виде контейнера. На следующем ниже рисунке показано, как YARN обрабатывает запросы от многочисленных клиентов.

Следующий ниже рисунок объясняет архитектуру YARN в Hadoop:



Менеджер ресурсов	Менеджер узла	Менеджер приложения
Управляет доступностью менеджера узла, отслеживает ведущий узел приложения	Обеспечивает ресурсами для выполнения вычислений	Координирует работу менеджера ресурсов и менеджера узлов при выделении ресурсов под задачу и их наличие

Настройка Hadoop в Linux

Для настройки Hadoop нам нужно скачать его с `apache.org` и сконфигурировать различные параметры для Hadoop. Для запуска служб Hadoop необходимо сконфигурировать различные параметры, такие как узел имен, узел данных и YARN.

1. Сначала нам нужно скачать Hadoop с `apache.org`:

```
[root@ip-172-31-22-59 local]# wget http://www-us.apache.org/dist/hadoop/common/h
adoop-2.6.5/hadoop-2.6.5.tar.gz
--2017-10-02 07:25:51--  http://www-us.apache.org/dist/hadoop/common/hadoop-2.6.
5/hadoop-2.6.5.tar.gz
Resolving www-us.apache.org (www-us.apache.org)... 140.211.11.105
Connecting to www-us.apache.org (www-us.apache.org)|140.211.11.105|:80... connec
ted.
HTTP request sent, awaiting response... 200 OK
Length: 199635269 (190M) [application/x-gzip]
Saving to: 'hadoop-2.6.5.tar.gz'

hadoop-2.6.5.tar.gz 100%[===================>] 190.39M  7.93MB/s    in 14s

2017-10-02 07:26:06 (13.3 MB/s) - 'hadoop-2.6.5.tar.gz' saved [199635269/1996352
69]

[root@ip-172-31-22-59 local]#
```

2. После установки Hadoop мы можем сконфигурировать переменные среды в системе. Также нам нужно настроить YARN на сервере. YARN является подразделом Hadoop, отдельной частью MapReduce по обработке и планированию данных и управлению ресурсами, с целью ускорения обработки данных

алгоритмом MapReduce. Детальное рассмотрение YARN выходит за рамки этой книги. Для настройки Apache Hadoop выполните следующие действия.

```
[root@ip-172-31-22-59 local]# mv hadoop-2.6.5 hadoop
[root@ip-172-31-22-59 local]# export HADOOP_HOME=/usr/local/hadoop
[root@ip-172-31-22-59 local]# export HADOOP_MAPRED_HOME=$HADOOP_HOME
[root@ip-172-31-22-59 local]# export HADOOP_COMMON_HOME=$HADOOP_HOME
[root@ip-172-31-22-59 local]# export HADOOP_HDFS_HOME=$HADOOP_HOME
[root@ip-172-31-22-59 local]# export YARN_HOME=$HADOOP_HOME
[root@ip-172-31-22-59 local]# export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
[root@ip-172-31-22-59 local]# export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
[root@ip-172-31-22-59 local]# source ~/.bashrc
[root@ip-172-31-22-59 local]# cd $HADOOP_HOME/etc/hadoop
```

Нам нужно задать путь к Java в конфигурации Hadoop, как продемонстрировано ниже:

```
cd $HADOOP_HOME/etc/hadoop
export JAVA_HOME=/usr/local/java
```

3. Как только параметры заданы, нужно обновить файлы конфигурации Hadoop, для того чтобы запустить в Hadoop.
4. В файле `core-site.xml` обновите свойство, как показано на следующем ниже снимке экрана; файл `core-site.xml` сообщает Hadoop, где на сервере будет выполняться узел имен:

```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://localhost:9000 </value>
   </property>
</configuration>
```

5. После того как файл `core-site.xml` задан, нам нужно задать узел имен и узел данных в Hadoop HDFS. Ниже показана конфигурация, требующаяся в файле `hdfs-site.xml` для настройки узла имен и узла данных:

```
<configuration>
<property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>

   <property>
      <name>dfs.name.dir</name>
      <value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>
   </property>

   <property>
      <name>dfs.data.dir</name>
      <value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>
   </property>
</configuration>
```

6. Нам нужно задать конфигурацию YARN на сервере, как показано далее:

```
-->
<configuration>

<!-- Site specific YARN configuration properties -->
<property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
</configuration>
```

7. Затем нам нужно установить файл конфигурации mapred. Файл `mapred-site.xml` поставляется по умолчанию с Hadoop с суффиксом `.template`. Сначала

нужно переименовать существующий файл из `mapred-site.xml.template` в `mapred-site.xml`, а затем его сконфигурировать, как показано далее:

```
cp mapred-site.xml.template mapredsite.xml
```

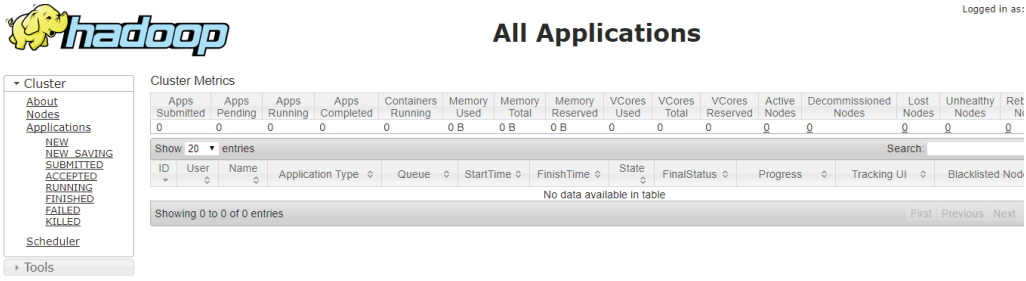
Ниже приводится содержимое файла `mapred-site.xml`:

```
<configuration>
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
   </property>
</configuration>
~
```

8. Теперь мы закончили с конфигурацией Hadoop. Запустите службы Hadoop, как показано в следующих ниже командах:

```
$ start-dfs.sh
$ start-yarn.sh
```

После запуска служб Hadoop мы можем проверить интерфейс приложения кластера Hadoop в браузере, обратившись к нему через порт 8088. Например, `http://localhost:80888` покажет список приложений, запущенных в кластере Hadoop:



Инсталляция Apache Sqoop

Теперь, когда мы закончили с Apache Hadoop, мы можем настроить Sqoop на сервере. Ниже приведены команды для настройки Sqoop на сервере:

```
[root@ip-172-31-22-59 local]# wget http://www-us.apache.org/dist/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
--2017-10-08 13:26:33--  http://www-us.apache.org/dist/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
Resolving www-us.apache.org (www-us.apache.org)... 140.211.11.105
Connecting to www-us.apache.org (www-us.apache.org)|140.211.11.105|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16870735 (16M) [application/x-gzip]
Saving to: 'sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz'

sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.g 100%[=====================================================================================>]  16.09M  13.6MB/s    in 1.2s

2017-10-08 13:26:34 (13.6 MB/s) - 'sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz' saved [16870735/16870735]

[root@ip-172-31-22-59 local]# tar xf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
[root@ip-172-31-22-59 local]# mv sqoop-1.4.6.bin__hadoop-2.0.4-alpha sqoop
[root@ip-172-31-22-59 local]#
```

После того как Sqoop будет загружен, мы должны активизировать его конфигурацию из каталога `conf`:

```
cd $SQOOP_HOME/conf
mv sqoop-env-template.sh sqoop-env.sh
```

Теперь нам нужно настроить Sqoop, указав пути установки Hadoop. На следующем ниже снимке экрана показано, как настроить это в Sqoop:

```
[root@ip-172-31-22-59 bin]# export HADOOP_COMMON_HOME=/usr/local/hadoop
[root@ip-172-31-22-59 bin]# export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

Apache Sqoop теперь настроен на нашем сервере. Мы можем настроить коннектор MySQL для Sqoop, и это будет одним из наших последних шагов в настройке Apache Sqoop с Hadoop и MySQL 8.

Конфигурирование коннектора MySQL

Теперь, когда Hadoop скачан на сервер и Sqoop тоже настроен, нам нужно установить коннектор MySQL для Sqoop для передачи данных из MySQL в Hadoop.

Коннектор MySQL можно скачать с `https://dev.mysql.com/downloads/connector/j/8.0.html`. После скачивания коннектора MySQL примените приведенные ниже команды, чтобы скопировать исполняемый файл `jar` коннектора MySQL в Sqoop:

```
unzip mysql-connector-java-5.1.19.zip
cd mysql-connector-java-5.1.19
mv mysql-connector-java-5.1.19-bin.jar /usr/local/sqoop/lib
```

Мы закончили со всеми конфигурациями, необходимыми для Apache Sqoop для импорта/экспорта данных между Hadoop и MySQL 8. В следующем разделе мы узнаем, как импортировать данные в Hadoop HDFS с помощью Apache Sqoop.

Импортирование неструктурированных данных в Hadoop HDFS из MySQL

С помощью Sqoop мы можем передавать данные из реляционной системы управления базами данных в Hadoop HDFS. Поскольку для подключения к источнику Sqoop использует драйвер Java Database Connectivity (JDBC), он может применяться с любой реляционной базой данных, имеющей поддержку строк подключения JDBC. В предыдущем разделе мы загрузили и настроили коннектор MySQL для Sqoop, поэтому теперь давайте посмотрим, как подключиться к базам данных MySQL из Sqoop и передавать данные в HDFS.

Импорт Sqoop для извлечения данных из MySQL 8

Чтобы разобраться в процессе импорта Sqoop, давайте создадим базу данных и таблицу в MySQL 8, которую мы будем использовать на протяжении всей главы для демонстрации примеров:

```
mysql> create database hadoop_bigdata;
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> CREATE TABLE
    ->  `hadoop_bigdata`.`users` (
    -> `user_id` INT NOT NULL AUTO_INCREMENT ,
    -> `email` VARCHAR(200) NOT NULL ,
    -> `date_of_joining` DATE NOT NULL ,
    -> `date_of_birth` DATE NOT NULL ,
    -> `first_name` VARCHAR(200) NOT NULL ,
    -> PRIMARY KEY (`user_id`)
    -> ) ENGINE = InnoDB;
Query OK, 0 rows affected (0.02 sec)

mysql>
```

Sqoop предоставляет команду `import` для импорта данных из реляционной базы данных в HDFS. Ниже приведены общие команды, используемые для импорта данных с помощью Sqoop:

```
sqoop import (общие-аргументы) (аргументы-импорта)
sqoop-import (общие-аргументы) (аргументы-импорта)
```

- `общие-аргументы` – это универсальные параметры экспорта, такие как предоставление строки подключения JDBC, имя драйвера JDBC, имя и пароль пользователя для подключения к базе данных;
- `аргументы-импорта` - это различные параметры, поддерживаемые Sqoop для импорта данных из реляционной базы данных.

Чтобы подключиться к базе данных, мы можем использовать следующую ниже команду:

```
sqoop import --driver {классДрайвераДляПодключения} --connect
{СтрокаПодключения}
```

Здесь `классДрайвераДляПодключения` – это имя класса, являющегося частью коннектора MySQL, который мы уже скачали. `СтрокаПодключения` – это параметр, который мы будем использовать для подключения к хосту MySQL с помощью драйвера `jdbc`.

Приведенная ниже команда подключит хост на `myhost.com` к серверу MySQL с базой пользователей `users`:

```
sqoop import --connect jdbc:mysql://myhost.com/hadoop_bigdata
```

Если подключение к MySQL требует аутентификации, то для подключения к требуемой базе данных в качестве входных параметров строки подключения нам нужно использовать имя и пароль пользователя:

```
sqoop import --connect jdbc:mysql://myhost.com/hadoop_bigdata --username
root --password mysql@123
```

Сейчас мы подключены к базе данных `hadoop_bigdata`. Теперь мы можем выбрать данные, которые должны быть реплицированы на HDFS.

Мы можем выбрать таблицу, которая должна быть реплицирована из таблицы данных `hadoop_big`:

```
sqoop import --connect jdbc:mysql://myhost.com/hadoop_bigdata --username
root --password mysql@123 --table users
```

В случае если нам нужно скопировать выбранные столбцы из таблицы `users`, мы можем указать список столбцов, которые должны быть скопированы из MySQL 8:

```
sqoop import --connect jdbc:mysql://myhost.com/hadoop_bigdata
--username root --password mysql@123
--table users
--columns "user_id,email,date_of_joining,date_of_birth"
```

В той же самой синтаксической конструкции мы можем также указать условие `--where`, чтобы скопировать базу данных из MySQL в HDFS:

```
sqoop import --connect jdbc:mysql://myhost.com/hadoop_bigdata --username
root --password mysql@123 --table users --where "date_of_birth >
'1979-12-31'"
```

Эта синтаксическая конструкция предпишет Sqoop скопировать все данные таблицы `users`, в которых дата рождения пользователя находится после 31 декабря 1979 года.

Вместо синтаксической конструкции для определения имени таблицы, столбца или условия `--where` в Sqoop можно также использовать запросы MySQL. В случае использования прямых запросов SQL для копирования данных мы должны указать конечный каталог, куда будут копироваться результаты запроса. Поскольку Sqoop выполняет параллельную обработку данных, к запросам, используемым для извлечения данных из исходной базы данных, он добавляет граничное условие и разделяющий столбец.

Если прямой запрос не используется, Sqoop автоматически определяет и добавляет граничные условия и разбивает их по фильтру перед запуском запроса. Разделяющий столбец обычно является первичным ключом таблицы. В случае использования прямых запросов мы должны предоставить условие `--split-by`, которое будет использоваться для разделения результатов запроса для обработки. Кроме того, в запрос `WHERE` мы должны включить статический токен `$CONDITIONS`. Sqoop заменит токен `$CONDITIONS` на граничное условие.

Ниже приведен пример использования прямого запроса для импорта:

```
sqoop import
--query 'SELECT * FROM users WHERE $CONDITIONS'
--split-by users.id
--target-dir /hduser/hadoop_bigdata/users/userdata
```

Приведенный выше запрос говорит сам за себя. Этот запрос получит все записи из таблицы `users`. Принятое Hadoop по умолчанию значение для параллельной операции равняется четырем. Это означает, что по умолчанию Hadoop будет использовать четыре параллельные операции для копирования данных из исходной базы данных в HDFS. Это значение настраивается и может быть изменено при использовании команды импорта с параметром `-m` или `--num-mappers`. Хотя увеличение значения параметра может повысить производительность операций копирования, перед установкой любого значения в качестве числа параллельных операций, которые могут выполняться, требуется правильная точная настройка параметра, исходя из конфигурации кластера, а также количества потоков, поддерживаемых реляционными СУБД.

Используя Sqoop, мы можем импортировать все таблицы из базы данных. Ниже приведены команды, применяемые для импорта всех таблиц из базы данных:

```
sqoop import-all-tables (общие-аргументы) (аргументы-импорта)
sqoop-import-all-tables (общие-аргументы) (аргументы-импорта)
```

`общие-аргументы` остаются такими же, как и в команде импорта Sqoop. Мы можем указать один или несколько столбцов, которые необходимо пропустить при импорте всей таблицы. Параметр `--exclude-tables` можно использовать с командой `import-all-tables` для указания списка таблиц, которые будут исключены из списка:

```
sqoop import-all-tables
--connect jdbc:mysql://localhost/hadoop_bigdata
--username root --password Pass@123
--exclude-tables login_history
```

Предыдущая команда импортирует все таблицы из базы данных hadoop_big, кроме logic_history. Если вы хотите исключить более одной таблицы базы данных, необходимо указать имена таблиц, разделенные запятыми.

Инкрементный импорт с использованием Sqoop

Поскольку мы работаем с большими данными, стандартной тенденцией является неуклонный рост данных в исходных таблицах. Sqoop предоставляет возможность импортировать подобные данные инкрементно, так что команда импорта может выполнять импорт из последних данных, импортированных из исходной базы данных. Ниже приведены три параметра для использования инкрементного импорта, которые можно использовать с командами импорта Sqoop:

--check-column (столбец)	Столбец, который необходимо указать, где Sqoop будет проверять идентификатор последних импортированных записей
--incremental (режим)	Это тип инкрементного импорта
--last-value (значение)	Sqoop будет использовать этот параметр для идентификации новых записей, вставленных в базу данных с момента последнего импорта данных из исходной таблицы

Как упоминалось в предыдущей таблице, инкрементный импорт Sqoop можно выполнять либо в режиме дополнения, либо в режиме последнего изменения. Когда инкрементный импорт выполняется в режиме дополнения, параметр --check-columns будет содержать имя столбца, в котором Sqoop будет сравнивать значение идентификатора, импортированное последним. Sqoop будет использовать параметр --last-value, чтобы узнать, какая строка была импортирована последней.

Если задан инкрементный режим последнего изменения, Sqoop будет сравнивать дату, указанную в --last-value, чтобы узнать последнее вставленное значение. И все данные, которые были изменены после даты, указанной в --last-value, будут импортированы в Hadoop HDFS.

Загрузка структурированных данных в MySQL с помощью Apache Sqoop

Таким образом, к настоящему времени мы импортировали все данные в HDFS, и данные были обработаны с помощью MapReduce. Теперь нам нужно перенести данные в MySQL. Apache Sqoop предоставляет функцию экспорта для экспорта данных из хранилища HDFS в реляционную базу данных, такую как MySQL.

Экспорт Sqoop для хранения структурированных данных в MySQL 8

Функция экспорта Apache Sqoop может передавать данные обратно в реляционную базу данных с помощью следующих ниже трех методов:

- в режиме вставки insert Sqoop будет генерировать запросы на вставку для данных, которые будут вставлены в MySQL. Вставка – это режим по умолчанию для передачи данных из HDFS с помощью Sqoop;
- в режиме обновления update Sqoop предоставляет инструкции обновления для обновления данных в существующих записях конечной базы данных;

- в режиме вызова call Sqoop будет выполнять вызов хранимой процедуры в итерации для данных в HDFS.

Ниже приведены универсальные команды, доступные для экспорта данных с помощью Sqoop:

```
sqoop export (общие-аргументы) (аргументы-экспорта)
sqoop-export (общие-аргументы) (аргументы-экспорта)
```

общие-аргументы, как мы узнали ранее в разделе импорта Sqoop, – это универсальные параметры для экспорта данных из HDFS, такие как строка подключения JDBC, имя драйвера JDBC, имя и пароль пользователя для подключения к базе данных.

аргументы-экспорта – это списки параметров, которые предоставляют различные способы экспорта данных в конечную базу данных. Ниже приведен список нескольких параметров, которые можно использовать при экспорте. Полный список аргументов-экспорта можно найти на sqoop.apache.org.

--columns	Определяет имена столбцов, которые необходимо экспортировать из HDFS. Имена столбцов должны передаваться через запятую
--export-dir имякаталога	Определяет исходный путь HDFS, откуда данные будут экспортироваться с помощью Sqoop
--num-mappers	Количество параллельных операций, разрешенных для экспорта данных
--table	Определяет имя таблицы, в которую будут помещены экспортированные данные
--call	Если экспорт выполняется с помощью процедуры сохранения MySQL, то для определения имени процедуры сохранения будет использоваться параметр --call
--update-key	Если экспорт выполняется путем замены существующих данных в целевой таблице, то для определения первичного ключа будет использоваться параметр --update-key. Sqoop идентифицирует существующие записи в целевой таблице с помощью параметра --update-key
--update-mode	Режим экспорта по умолчанию заключается в добавлении новых данных в конец таблицы. Если экспорт выполняется путем замены существующих данных в целевой таблице, то нам необходимо явно определить режим экспорта как режим обновления

В Sqoop экспорт данных из HDFS осуществляется параллельно. Каждый экспорт выполняется методом массовой вставки. Каждый из процессов экспорта Sqoop создает один запрос на массовую вставку из 100 записей. Операция экспорта использует транзакции для импорта данных экспорта. Команда экспорта фиксирует данные в целевой таблице после каждых 100 запросов INSERT, и каждый запрос на массовую вставку содержит 100 записей; это означает, что одна операция вставки фиксируется в базе данных после каждых 10K записей.

Давайте разберемся с командой экспорта на примере. В разделе «*Импорт Sqoop для извлечения данных из MySQL 8*» мы импортировали некоторые данные из таблицы users. Давайте экспортируем эти же данные назад в таблицу users, используя команду экспорта:

```
sqoop export
--connect jdbc:mysql://myhost.com/hadoop_bigdata
--username root --password mysql@123
--table users
--export-dir /hduser/hadoop_bigdata/users/userdata
```

Если нам нужно экспортировать фиксированное количество столбцов, то в качестве альтернативы мы можем использовать следующую ниже команду:

```
sqoop export
--connect jdbc:mysql://myhost.com/hadoop_bigdata
--username root --password mysql@123 --table users
--columns "col1,col2,col3"
--export-dir /hduser/hadoop_bigdata/users/userdata
```

Сохраненные задания Sqoop

Импорт и экспорт данных, которые мы пошагово изучали ранее, могут храниться в виде заданий, которые могут повторяться несколько раз для выполнения одной и той же операции снова и снова. Сохраненное задание также можно настроить для выполнения операции инкрементного импорта. В таких случаях информация о последних импортированных строках хранится в Sqoop, что помогает Sqoop перезапустить процесс импорта с того места, где он был остановлен в последний раз. Ниже приведена синтаксическая конструкция для использования команды Sqoop по выполнению заданий для сохранения операций импорта и экспорта:

```
sqoop job (общие-аргументы)
          (аргументы_задания)
          [-- [имя-подынструмента] (аргументы-подинструмента)]
sqoop-job (общие-аргументы)
          (аргументы_задания)
          [-- [имя-подынструмента] (аргументы-подинструмента)]
```

- `общие-аргументы` дает возможность создавать новые задания, удалять существующие задания, перечислять все существующие задания или выполнять созданные задания;
- `аргументы_задания` – это имя задания, для которого мы хотим выполнить команду `job`. Это может быть имя задания, которое нам нужно создать, показать, удалить или выполнить;
- `имя-подынструмента` – это имя операции, для которой мы хотим создать задание. Это может быть импорт или экспорт;
- `аргументы-подынструмента` - это параметры, которые мы используем для импорта или экспорта данных при выполнении задания.

Рассмотрим следующий ниже пример, где мы создаем задание для импорта данных из таблицы `users`:

```
sqoop job
--create userimport
import --connect jdbc:mysql://myhost.com/hadoop_bigdata
--username root --password mysql@123
--table users
```

Точно так же мы можем создать задание для экспорта данных из таблицы `users`:

```
sqoop job
--create userexport
export --connect jdbc:mysql://myhost.com/hadoop_bigdata
--username root --password mysql@123
--table users
--columns "col1,col2,col3"
--export-dir /hduser/hadoop_bigdata/users/userdata
```

Точно так же мы можем перечислить все существующие задания, используя следующую ниже команду:

```
sqoop job --list
```

Или же, чтобы выполнить существующее задание, мы можем запустить следующую ниже команду `sqoop`, которая запустит выполнение процесса:

```
sqoop job --exec userimport
```

Исполнив приведенную выше команду, мы можем выполнить импорт сохраненного задания `userimport`. Таким образом, без написания полной синтаксической конструкции команды импорта или экспорта снова и снова мы можем легко запускать их с помощью сохраненных заданий.

Резюме

В этой главе мы выяснили, каким образом анализ журналов операций способен помочь магазину электронной коммерции повлиять на увеличение объема продаж и повышение опыта пользователя, используя информацию журнала посетителей, что в конечном счете увеличит прибыль от деятельности предприятия. Мы узнали, как MySQL 8 и Hadoop могут использоваться для создания отчетов о поведении пользователей. Мы рассмотрели, как Sqoop может быть очень полезен в обмене данными между MySQL 8 и Hadoop HDFS. Мы дали краткое объяснение функционала Hadoop, мы подробно рассмотрели Apache Sqoop и узнали, как использовать операции импорта и экспорта Sqoop.

В следующей главе мы узнаем, что такое обработка данных в режиме реального времени и как механизм репликации событий MySQL Applier может использоваться для обработки данных в режиме реального времени.

Глава 10

Практический пример: часть II. Обработка событий в режиме реального времени с помощью MySQL Applier

В предыдущей главе мы рассмотрели Hadoop, а также установку Hadoop в среде Linux. Мы также познакомились с работой Apache Sqoop по интеграции Hadoop с MySQL 8 для импорта/экспорта данных с помощью пакетной обработки. Мы обсудили практический пример использования Apache Sqoop из реальной жизни.

В этой главе мы поговорим о механизме репликации событий MySQL Applier, который будем использовать для импорта данных из MySQL 8 в Hadoop с использованием обработки в режиме реального времени. Мы рассмотрим следующие темы:

- обзор практического примера;
- обзор механизма MySQL Applier;
- интеграция в режиме реального времени с помощью MySQL Applier;
- организация и анализ данных в Hadoop.

Обзор практического примера

СУБД MySQL является проверенным решением для хранения транзакционных данных, которые используются для поддержания свойств ACID во время операций записи. Начиная с MySQL 5.6 она также включает новый интерфейс `NoSQL Memcached` для подсистемы хранения данных InnoDB, который улучшает производительность для усвоения крупных объемов данных. Hadoop используется для хранения огромного количества данных (петабайты) и их обработки для многих сценариев, таких как хранение архивных данных или различных исторических данных. Аналитическая обработка данных осуществлялась в автономном режиме и не являлась составной частью обработки данных. Тем не менее технология эволюционировала и в настоящее время Hadoop является активной частью потоков данных для многих случаев использования, когда мы требуем обработки и предоставления данных пользователю в режиме реального времени.

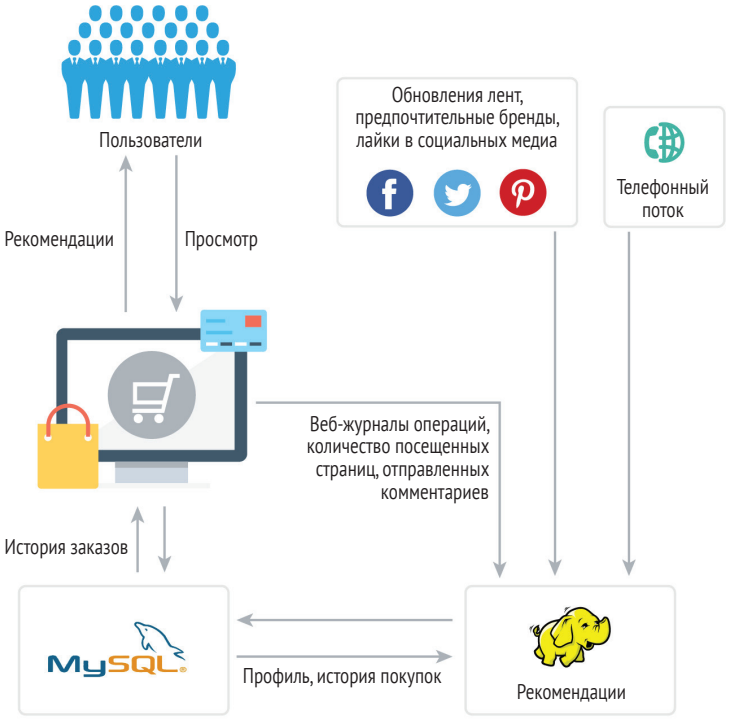
Мы можем использовать MySQL для хранения транзакционных данных и Hadoop для хранения огромного количества данных, которые могут легко обраба-

тываться с помощью алгоритма MapReduce. Мы можем воспользоваться преимуществами обеих технологий, чтобы снять препятствия для анализа больших данных. Имеются самые разные случаи применения, когда нам нужно использовать интеграцию с Hadoop и MySQL в режиме реального времени:

- рекомендательный механизм в электронной коммерции;
- анализ мнений;
- анализ маркетинговых кампаний;
- обнаружение мошенничества.

В нашем случае давайте обсудим обработку данных в режиме реального времени, чтобы разработать рекомендательный механизм для клиентов, когда они посещают портал электронной коммерции. Клиенты покупают товары на портале электронной коммерции, которые хранятся в истории заказов клиента. Во время покупки клиент переходит с портала электронной коммерции на другие страницы; такая история просмотра должна быть сохранена, так как она будет необходима для дальнейшего анализа. Предпочтения клиентов, в частности предпочтительные бренды на различных социально-медийных платформах, таких как Facebook, Google Plus, Twitter и т. д., также должны быть извлечены для анализа. Основываясь на истории заказов, истории просмотров или данных социальных медиа, вы можете разработать алгоритм отображения рекомендуемых товаров клиентам.

Обработка данных должна выполняться в режиме реального времени для получения точного результата рекомендации и достижения наилучшего результата с точки зрения продажи того, что клиенты предпочитают в магазине электронной коммерции. Ниже приводится наглядное представление рабочего процесса, на основе которого будут делаться рекомендации.



Hadoop является одной из лучших платформ для выполнения анализа посещаемости сайтов пользователями, данных социальных сетей, истории заказов и т. д. Мы можем использовать решение электронной коммерции, такое как Magento, в качестве портала электронной коммерции. Портал электронной коммерции использует базу данных MySQL для хранения каталога, сведений о клиентах, истории заказов и многого другого. Hadoop может использоваться для хранения данных перемещения по страницам и предпочтений в социальных медиа. Для извлечения данных социальных медиа имеются различные `API`, которые могут использоваться для извлечения брендовых предпочтений клиента. Используя процесс Hadoop MapReduce, мы можем разработать рекомендательный механизм, где Hadoop обрабатывает историю заказов, просматривает данные журналов операций и социальных медиа для определения рекомендуемых клиенту товаров.

Но тогда каким образом передавать данные между MySQL и Hadoop для выполнения анализа в режиме реального времени? При интеграции MySQL с Hadoop нам нужно будет передавать захваченные данные из MySQL в Hadoop и наоборот, с тем чтобы реализовать аналитическую обработку на больших данных. Данные, существующие в MySQL, должны быть перемещены в **распределенную файловую систему Hadoop** (HDFS), в которой будет получен результат аналитической обработки, и эти аналитические результаты должны быть перемещены обратно в MySQL из Hadoop. Существует целый ряд способов перемещения данных между MySQL и Hadoop. Давайте посмотрим, как этого можно достичь и каким будет эффективный способ.

Для дампа и загрузки данных из MySQL 8 в Hadoop существует несколько решений и технологий. Ниже мы кратко рассмотрим некоторые из них.

MySQL Applier

Для интеграции данных в реальном времени между MySQL и Hadoop мы можем использовать механизм репликации событий MySQL Applier. Механизм репликации событий MySQL Applier работает как посредник между MySQL 8 и Hadoop для репликации данных. Applier считывает событие двоичного журнала, декодирует строки в формат и применяет эти события к таблице Hive.

Дамп и импорт SQL

В этом процессе мы должны написать пользовательский сценарий для извлечения данных из MySQL в формате, который мы можем импортировать в Hadoop с помощью сценария. Этот процесс опирается на существующие инструменты базы данных для извлечения данных.

Давайте посмотрим на поток импорта и экспорта данных с помощью этого метода. Например, у нас есть таблица пользователей `users` со структурой данных, показанной ниже. Мы экспортируем эту таблицу пользователей из MySQL и импортируем в Hadoop.

user_id	first_name	last_name	email
1	Chintan	Patel	chintan.patel@gmail.com
2	Jaydeep	sharman	jaydeep.sharman@gmail.com

Для экспорта данных в CSV-файл можно применить следующий ниже запрос:

```
SELECT user_id, first_name, last_name, email into OUTFILE 'users.csv'
FIELDS TERMINATED BY ',' FROM users;
```

Скопируйте этот файл на сервер Hadoop, чтобы загрузить его в Hadoop с помощью следующей ниже команды:

```
$ hdfs dfs mkdir users
$ hdfs dfs -copyFromLocal users.csv users
```

Это ручной процесс импортирования таблицы пользователей как `.csv`-файл и импортирования того же файла в Hadoop. Преимуществом этого решения является гибкость – мы можем создавать пользовательские форматы для извлечения данных, и репликация не требуется. Вместе с тем будут накладные расходы, связанные с запросом к базе данных для экспорта, который должен рассматриваться для крупного объема данных.

В случае экспорта/импорта крупного объема данных этот метод занимает больше времени. Кроме того, эта файловая операция будет потреблять больше ресурсов, что вызывает снижение производительности в других областях.

Sqoop

Apache Sqoop используется для эффективной передачи крупного объема данных между Apache Hadoop и реляционными системами управления базами данных, такими как MySQL, Oracle и Postgres. Sqoop может импортировать или экспортировать данные параллельно, тем самым ускоряя передачу данных между исходной и конечной базами данных. Sqoop обеспечивает пакетную обработку для обмена данными между MySQL и Hadoop. Мы обсудили этот метод в главе 9 «*Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop*».

Преимущество использования Apache Sqoop заключается в том, что он позволяет гибко выбирать данные, автоматизировать передачу и не требует активировать репликацию на MySQL.

Ограничение Apache Sqoop заключается в том, что он использует пакетную передачу, поэтому нам придется передавать данные каждый раз, когда мы хотим обновить базу данных. Это создаст накладные расходы для операционной системы и памяти, что влияет на производительность. Кроме того, это неэффективный способ передачи крупного объема данных в случае, когда требуется обновить только несколько изменений. На обновление нескольких изменений уходит много времени.

Репликатор Tungsten

Tungsten replicator – это механизм репликации с открытым исходным кодом, который работает с несколькими различными исходными и конечными базами данных. Эта функциональность допускает репликацию между MySQL, Oracle, Hadoop в числе прочих.

Apache Kafka

Распределенная потоковая платформа Apache Kafka используется для построения конвейеров потоковой передачи данных в режиме реального времени, которые надежно получают данные между системами или приложениями.

Talend

Talend – это инструмент ETL, применяемый для передачи данных из различных источников в многочисленные конечные пункты. Используя это решение, мы можем применять коннектор Sqoop для создания упрощенного интерфейса извлечения и миграции. Talend имеет такие же ограничения, как и те, что касаются Sqoop.

Dell Shareplex

Коммерческий инструмент, который обеспечивает репликацию данных из архивных журналов Oracle в Hadoop практически в режиме реального времени.

Сравнение инструментов

До сих пор мы видели в основном три решения для достижения экспорта/импорта между MySQL и Hadoop. Каждый инструмент имеет свои плюсы и минусы. Давайте сравним эти три решения, чтобы определить, какое из них подходящее для наших потребностей.

	Дамп и импорт SQL	**Sqoop**	**MySQL Applier**
Процесс	Ручной/сценарный	Ручной/сценарный	Полностью автоматизированный
Инкрементная загрузка	Возможна с изменениями **языка определения данных** (DDL)	Требуются изменения **языка определения данных** (DDL)	Полностью поддерживается
Задержка	Высокая нагрузка	Периодическая	В реальном времени
Потребность в извлечении	Полный скан таблицы	Полный и частичный сканы таблицы	Скан двоичного журнала с низким воздействием

В сущности, метод *дампа и импорта SQL* используется редко и неосуществим для крупных наборов данных. Чтобы преодолеть ручное управление и долгое время, Apache Sqoop может помочь, предоставляя лучший автоматический способ передачи данных между Hadoop и MySQL.

Но, как мы видели, Apache Sqoop также загружает основную массу данных, а не сами изменения. Это будет занимать больше времени в зависимости от размера данных. Именно здесь вступает в игру обработка данных в режиме реального времени, имея в виду автоматическую загрузку данных, когда происходят новые изменения, и их обновление вместо повторной загрузки всех данных. MySQL 8 имеет возможности обрабатывать такие данные в режиме реального времени с помощью механизма репликации событий MySQL Applier. Давайте проведем краткий обзор MySQL Applier, который помогает в обработке данных в режиме реального времени.

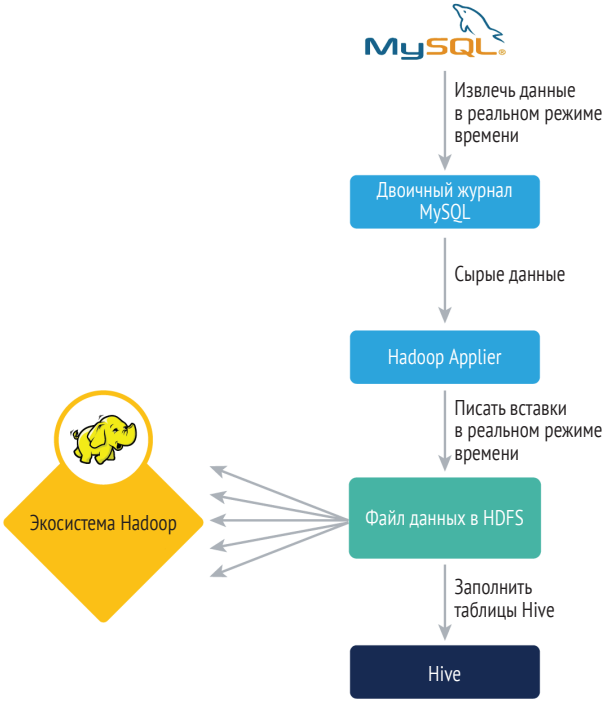
Обзор MySQL Applier

MySQL Applier обеспечивает обработку данных в режиме реального времени между MySQL и Hadoop. Это довольно эффективный способ загрузки данных ввиду его производительности и выполнения в режиме реального времени. Вместо загрузки всех данных он позволяет загружать только измененные данные. Следовательно, **больше не требуется массовая передача данных**!

MySQL Applier реплицирует строки, вставленные в MySQL, в HDFS с помощью двоичного журнала MySQL. Он будет использовать двоичный журнал и вставлять данные в режиме реального времени, основываясь на событиях в MySQL. Все события, произошедшие в MySQL Server, доступны в двоичном журнале, и MySQL Applier принимает эти изменения из событий и применяет те же самые в Hadoop. Благодаря этому мы можем быстро получать новые данные из MySQL.

MySQL Applier использует API, предоставляемый C-библиотекой `libhdfs`. Эта библиотека предварительно скомпилирована с дистрибутивами Hadoop для подключения к ведущему серверу MySQL или чтения файла двоичного журнала MySQL. Она отвечает за различные операции, перечисленные в следующем ниже списке:

- получить события (вставка, обновление, удаление), произошедшие на сервере MySQL;
- декодировать события, читая двоичный журнал и извлекая данные, которые используются событиями для внесения изменений в Hadoop;
- использовать обработчики содержимого, чтобы получить его в требуемом формате, и дописать его в текстовый файл в HDFS.



Приведенная выше схема объясняет то, как развивается процесс при интеграции с MySQL Applier. Любая операция, выполняемая на сервере MySQL, поступает в двоичный журнал. Он будет иметь табличную сырую информацию, которая была изменена или добавлена. MySQL Applier берет эту сырую информацию и немедленно применяет изменения в HDFS.

С помощью MySQL Applier мы имеем взаимодействие в режиме реального времени между MySQL и Hadoop, которое решает многие случаи применения ана-

литической обработки больших данных, в частности сбытовые кампании, анализ мнений, анализ журналов операций и т. д.

Инсталляция MySQL Applier

Теперь давайте рассмотрим установку и настройку механизма репликации событий MySQL Applier для обработки данных в режиме реального времени.

Ниже приведены предварительные требования для MySQL Applier:

- пакет MySQL Applier;
- Hadoop;
- Java;
- `libhdfs`;
- `cmake`;
- `libmysqlclient`;
- компилятор `gcc`, который требуется для выполнения программы C++;
- Сервер MySQL;
- `FindHDFS.cmake`;
- Hive.

Некоторые предварительные требования, такие как Hadoop, Java, установка сервера MySQL и клиента MySQL, уже обсуждались в главе 9 «*Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop*». Давайте взглянем на установку остальных предварительно необходимых компонентов.

libhdfs

Это прекомпилированные библиотеки с Hadoop. Вы можете найти файл `libhdfs.so` по пути `[Hadoop]/lib/native`. Для этой библиотеки мы должны экспортировать переменную среды, используя следующую ниже команду:

```
export HDFS_LIBS=/usr/local/hadoop/lib/native
```

cmake

`cmake` – это кросс-платформенная система сборки. Вы можете создать простой файл конфигурации в исходном каталоге `CMakeLists.txt`. Для установки на сервер вы можете скачать последнюю версию с https://cmake.org/download/. Выполните следующую ниже команду из терминала Linux для установки:

```
$ wget https://cmake.org/files/v3.9/cmake-3.9.4.tar.gz
$ tar xzf cmake-3.9.4.tar.gz
$ cd cmake-3.9.4.tar.gz
$ ./configure --prefix=/opt/cmake
$ make
$ make install
$ export PATH=/opt/cmake/bin/
```

Проверить установку можно с помощью такой команды:

```
cmake -version
```

Возвращает версию установленного `cmake`.

gcc

Компилятор gcc используется для запуска программы C++. Его можно установить, выполнив следующие ниже действия:

```
$ sudo yum install gcc
```

Проверить установку можно с помощью команды

```
$ gcc --version
```

FindHDFS.cmake

Найдите файл HDFS.cmake, чтобы найти библиотеку libhdfs при компиляции. Вы можете скачать этот файл с https://github.com/cloudera/Impala/blob/master/cmake_modules/FindHDFS.cmake. После загрузки экспортируйте переменную CMAKE_MODULE_PATH с помощью следующей ниже команды:

```
$ export CMAKE_MODULE_PATH=/usr/local/MySQL-replicationlistener/FindHDFS.cmake
```

Hive

Hive – это инфраструктура хранилища данных, построенная на базе платформы Hadoop и использующая ее модель хранения и исполнения. Эта инфраструктура была первоначально разработана компанией Facebook. Она предоставляет язык запросов, похожий на SQL и известный как **язык запросов Hive** (HiveQL). Используя этот язык, мы можем анализировать крупные наборы данных, хранящиеся в файловых системах, таких как HDFS. Hive также предоставляет функционал индексирования. Он предназначен для поддержки ситуативных запросов и легкого суммирования данных, а также для анализа крупных объемов данных.

Таблицы Hive аналогичны таблицам в реляционной базе данных, состоящим из разделов. Мы можем использовать HiveQL для доступа к данным. Таблицы сериализуются и имеют соответствующий каталог HDFS в базе данных. Hive позволяет исследовать и структурировать эти данные, анализировать их, а затем превращать в деловые идеи.

Чтобы установить Hive на сервер, загрузите пакет с archive.apache.org и извлеките его в домашний каталог Hadoop:

```
$ cd /usr/local/hadoop
$ wget
http://archive.apache.org/dist/hive/hive-0.12.0/hive-0.12.0-bin.tar.gz
$ tar xzf hive-0.12.0-bin.tar.gz
$ mv hive-0.12.0-bin hive
```

Чтобы создать каталог Hive в Hadoop и дать разрешение папке, исполните следующие ниже команды:

```
$ cd /usr/local/hadoop/hive
$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$ $HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

Переменные среды можно настроить с помощью следующей ниже команды:

```
$ export HIVE_HOME="/usr/local/hadoop/hive"
$ PATH=$PATH:$HIVE_HOME/bin
$ export PATHb
```

Чтобы загрузить терминал Hive, просто используйте команду hive:

```
[root@ip-172-31-22-59 hive]# hive
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.input.dir.recursive is deprecated. Instead, use mapreduce.input.fileinputformat.input.dir.recursive
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.max.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.maxsize
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.min.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.min.split.size.per.rack is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize.per.rack
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.min.split.size.per.node is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize.per.node
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
17/10/11 01:56:27 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative

Logging initialized using configuration in jar:file:/usr/local/hadoop/hive/lib/hive-common-0.12.0.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/hive/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hive>
```

Теперь можно использовать команду SQL для создания каталогов и документов, как описано в следующем ниже фрагменте кода:

```
hive> CREATE TABLE user (id int, name string);
OK
```

Интеграция в режиме реального времени с MySQL Applier

На GitHub имеется много пакетов приложений MySQL. Мы можем использовать любой из них, который обеспечивает основу для репликации и пример репликации в режиме реального времени:

- Flipkart/MySQL-replication-listener;
- SponsorPay/MySQL-replication-listener;
- bullsoft/MySQL-replication-listener.

Для нашей конфигурации давайте использовать Flipkart/MySQL-replication-listener. Вы можете клонировать библиотеку Git, используя следующую ниже команду:

```
$ git clone https://github.com/Flipkart/MySQL-replication-listener.git
```

Ниже приведены некоторые переменные среды, необходимые для пакета. Убедитесь, что все из них настроены правильно.

- `HADOOP_HOME`: путь к корневому каталогу Hadoop;
- `CMAKE_MODULE_PATH`: путь к корневому каталогу, в котором находятся файлы `FindHDFS.cmake`, и файлы `FindJNI.cmake` находятся в HDFS;
- `HDFS_LIB_PATHS`: путь файла `libhdfs.so` в Hadoop;
- `JAVA_HOME`: для этой переменной вам нужно установить домашний путь Java.

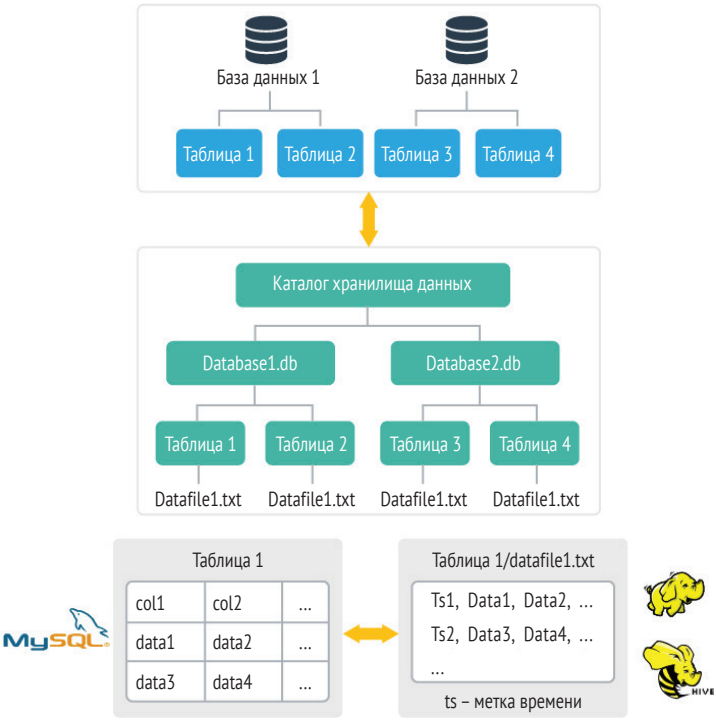
Теперь соберите и скомпилируйте все библиотеки, используя следующую ниже команду:

```
$ cd src
$ cmake . -DCMAKE_MODULE_PATH:String=/usr/local/cmake-3.10.0-rc1/Modules
$ make -j8
```

Пакет, сгенерированный из приведенной выше команды, будет использоваться для настройки репликации из MySQL 8 в Hadoop. Скомпилировав этот пакет, мы получим исполняемую команду `happlier`, которая будет использоваться для запуска репликации:

```
$ cd examples/mysql2hdfs/
$ cmake .
$ make -j8
```

Теперь, прежде чем начать репликацию, мы должны с помощью следующего ниже рисунка понять, как мы увязываем структуры данных MySQL и Hadoop.



Приведенная выше схема объясняет преобразование структуры данных для MySQL 8 и Hadoop. В Hadoop данные хранятся в виде файла данных. Механизму MySQL Applier не разрешается выполнять инструкцию DLL, поэтому мы должны создать базу данных и таблицу с обеих сторон. Для MySQL мы можем запустить инструкцию SQL для создания (CREATE) таблицы, в то время как в Hadoop мы можем использовать HIVE для создания базы данных и таблицы.

Ниже приведен SQL-запрос для создания таблицы, которую мы должны выполнить на сервере MySQL:

```
CREATE TABLE chintantable (i INT);
```

Ниже приведен запрос Hive для создания таблицы, которую мы должны выполнить из командной строки HIVE:

```
CREATE TABLE chintantable ( time_stamp INT, i INT) [ROW FORMAT DELIMITED]
STORED AS TEXTFILE;
```

После того как вы создадите базу данных и таблицу на MySQL и Hive, используйте следующую ниже команду, чтобы начать репликацию:

```
./happlier mysql://root@127.0.0.1:3306 hdfs://localhost:8088
```

Детали MySQL в команде `happlier` необязательны. По умолчанию в ней используется `mysql://user@127.0.0.1:3306`, и для HDFS детали конфигурируются в файле `coresite/core-default.xml`. Теперь для каждой строки, добавленной в базу данных MySQL, будет создана соответствующая строка в HDFS.

Когда любая операция вставки выполняется в базе данных MySQL, соответствующая ей строка реплицируется в таблице Hadoop. Мы также можем создать API для репликации операций обновления и удаления из `binlog API`.

Организация и анализ данных в Hadoop

Как мы узнали в главе 9 «*Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop*», Hadoop может использоваться для обработки неструктурированных данных, генерируемых посредством реляционных систем управления базами данных, таких как MySQL. В этом разделе мы узнаем, каким образом можно использовать Hadoop для анализа неструктурированных данных, генерируемых в MySQL 8. Основываясь на нашем примере магазина электронной коммерции, мы постараемся найти максимально продающийся товар среди клиентов, основываясь на истории заказов клиентов в магазине электронной коммерции. Мы перенесем данные заказов, сгенерированные в MySQL 8, в Apache Hive с помощью MySQL Applier. Затем мы будем использовать **язык запросов Hive** (Hive-QL) для анализа необходимых данных.

HiveQL применяет алгоритм MapReduce, который позволяет намного быстрее анализировать миллионы элементов данных за считанные секунды. Данные, генерируемые в Hive, могут быть переданы обратно в MySQL 8 как плоская таблица.

Рассмотрим следующую ниже таблицу истории заказов пользователя, сгенерированную в MySQL 8:

```
CREATE TABLE IF NOT EXISTS `orderHistory` (
 `orderId` INT(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
 `customerName` VARCHAR (100) NOT NULL,
 `customerBirthDate` DATE NULL,
 `customerCountry` VARCHAR(50),
 `orderDate` DATE,
 `orderItemName` VARCHAR (100),
 `orderQuantity` DECIMAL(10,2),
 `orderTotal` DECIMAL(10,2),
 `orderStatus` CHAR(2)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='USER ORDER HISTORY
information' AUTO_INCREMENT=1;
```

В этой таблице хранится информация об имени клиента, дате его рождения, регионе, а также информация о заказе, такая как дата заказа, название товара, количество, цена и состояние заказа.

Давайте вставим в эту таблицу некоторые примеры данных, которые мы будем использовать для передачи в Apache Hive с помощью следующего ниже примера:

```
mysql> INSERT INTO orderHistory (orderId, customerName, customerBirthDate, customerCountry,
orderDate, orderItemName, orderQuantity, orderTotal, orderStatus)
VALUES (111, "Jaydip", "1990-08-06", "USA", "2017-08-06", "Chair", 1, 500, 1);
mysql> INSERT INTO orderHistory (orderId, customerName, customerBirthDate, customerCountry,
orderDate, orderItemName, orderQuantity, orderTotal, orderStatus)
VALUES (222, "Shabbir", "1985-02-10", "India", "2017-09-06", "Table", 3, 1200, 1);
```

```
mysql> INSERT INTO orderHistory (orderId, customerName, customerBirthDate, customerCountry,
orderDate, orderItemName, orderQuantity, orderTotal, orderStatus)
VALUES (333, "Kandarp", "1987-04-15", "India", "2017-09-06", "Computer", 1, 43000, 1);
```

Ниже приведен результат из таблицы истории заказов `orderHistory`, который будет использоваться для дальнейшего анализа:

```
mysql> select * from orderHistory\G;
*************************** 1. row ***************************
 orderId: 111
 customerName: Jaydip
customerBirthDate: 1990-08-06
 customerCountry: USA
 orderDate: 2017-08-06
 orderItemName: Chair
 orderQuantity: 1.00
 orderTotal: 500.00
 orderStatus: 1
*************************** 2. row ***************************
 orderId: 222
 customerName: Shabbir
customerBirthDate: 1985-02-10
 customerCountry: India
 orderDate: 2017-09-06
 orderItemName: Table
 orderQuantity: 3.00
 orderTotal: 1200.00
 orderStatus: 1
*************************** 3. row ***************************
 orderId: 333
 customerName: Kandarp
customerBirthDate: 1987-04-15
 customerCountry: India
 orderDate: 2017-09-06
 orderItemName: Computer
 orderQuantity: 1.00
 orderTotal: 43000.00
 orderStatus: 1
3 rows in set (0.00 sec)
```

Теперь, прежде чем мы перенесем данные из MySQL в Hive, давайте создадим аналогичную схему в Apache Hive.

Ниже приведен пример создания таблицы в Hive:

```
CREATE TABLE orderHistory (
 orderId INT,
 customerName STRING,
 customerBirthDate DATE,
 customerCountry STRING,
 orderedDate DATE,
 orderItemName STRING,
 orderQuantity DECIMAL(10,2),
 orderTotal DECIMAL(10,2),
 orderStatus CHAR(2)
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

С помощью этого шага мы получили таблицу истории заказов `orderHistory` MySQL с данными, которые должны быть переданы, и таблицу Apache Hive `orderHistory`, которая готова получить входные данные. Начнем перенос данных из MySQL в Hive с помощью MySQL Applier.

Следующая ниже команда запустит MySQL Applier и начнет передачу данных из MySQL в Hive.

```
./happlier mysql://root@127.0.0.1:3306 hdfs://localhost:8088
```

У нас будут все строки таблицы истории заказов в Apache Hive. Мы можем использовать HiveQL для получения истории заказов максимально продающихся товаров. Следующим ниже является запрос на получение максимально продающегося товара:

```
SELECT
 orderItemName,SUM(orderQuantity),SUM(orderTotal)
FROM
 orderHistory
GROUP BY orderQuantity;
```

Этот запрос даст сумму количеств проданных товаров и их общую отпускную цену. Результат этих данных может храниться в текстовых файлах с разделителями-запятыми. Теперь эти текстовые файлы можно экспортировать обратно в MySQL с помощью Apache Sqoop. Мы познакомились с Apache Sqoop в главе 9 «*Практический пример: часть I. Apache Sqoop для обмена данными между MySQL и платформой Hadoop*».

Результат, произведенный для самого продаваемого товара в Hive, может быть экспортирован в плоскую таблицу в MySQL 8, которую можно легко использовать для отображения самого продаваемого товара. Аналогичным образом мы можем использовать таблицу истории заказов для создания других отчетов, таких как:

- наиболее продаваемые товары в различной возрастной группе;
- наиболее продаваемые товары по регионам;
- наиболее продаваемые товары по месяцам.

История заказов – это часть действий клиента в приложении электронной коммерции. Кроме нее, есть много других действий, таких как распространение в социальных медиа, закладки, упоминания, которые мы можем использовать для создания сильного рекомендательного механизма с использованием огромного количества генерируемых данных. Именно здесь вы можете использовать силу MySQL для больших данных!

Резюме

В этой главе мы рассмотрели практический пример рекомендательного механизма в приложении электронной коммерции. Мы выяснили различные инструменты для передачи данных из MySQL в технологии больших данных, такие как Hadoop. Мы изучили захватывающую тему механизма репликации событий MySQL Applier вместе с его инсталляцией и интеграцией. Затем мы разобрались с тем, как использовать MySQL Applier для обработки данных в режиме реального времени. Мы также научились организовывать и анализировать данные в Hadoop Hive и передавать данные из MySQL в Hadoop, используя MySQL Applier.

Предметный указатель

О

П

P

C

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Шаббир Чаллавала, Джадип Лакхатария,
Чинтан Мехта, Кандарп Патель

MySQL 8 для больших данных

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Логунов А. В.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 21,1875. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**