ГЛАВА 1

Введение в технологию Delta Lake для озер-хранилищ данных

В этой главе описывается история возникновения технологии Delta Lake и то, как она изначально разрабатывалась для решения проблем целостности данных в системах петабайтного масштаба. Если вы знакомы с историей Delta Lake и хотите перейти к ее описанию, структуре и протоколу транзакций Delta, сразу обратитесь к разделу «Что такое Delta Lake?» далее на с. 31.

Генезис Delta Lake

В этом разделе рассматривается краткая история развития Delta Lake: от возникновения идеи до его широкого признания сообществом в качестве формата для озер-хранилищ данных (lakehouse), обеспечивающего целостность важнейшего актива любого предприятия — его данных. Формат Delta Lake для озер-хранилищ данных был разработан, чтобы устранить ограничения, присутствующие в традиционных озерах и хранилищах данных. Он предоставляет ACID-транзакции (ACID — atomicity, consistency, isolation, durability — атомарность, согласованность, изолированность, долговечность) (https://oreil.ly/gjn55) и масштабируемую обработку метаданных, а также объединяет на одной платформе различные задачи аналитики данных, такие как пакетная и потоковая передача и обработка данных, машинное обучение и SQL.

Хранилища данных, озера данных и озера-хранилища данных

За последнее время в области информационных систем произошло много технологических прорывов (например, в высокопроизводительных вычислениях (HPC, high-performance computing) и в объектно-ориентированных базах данных); краткий обзор прогресса за последние несколько десятилетий в области обработки запросов и агрегации больших объемов бизнес-данных в информационных системах включает в себя следующие тематики: хранилища данных, озера данных и озера-хранилища. В целом все эти системы предназначены для оперативной аналитической обработки данных (OLAP, online analytics processing).

Хранилища данных

Хранилища данных специально разработаны для быстрого агрегирования и обработки больших объемов структурированных данных (рис. 1.1). Для их защиты обычно используются реляционные базы данных в целях осуществления ACID-транзакций, что имеет решающее значение для обеспечения целостности данных в бизнес-приложениях.

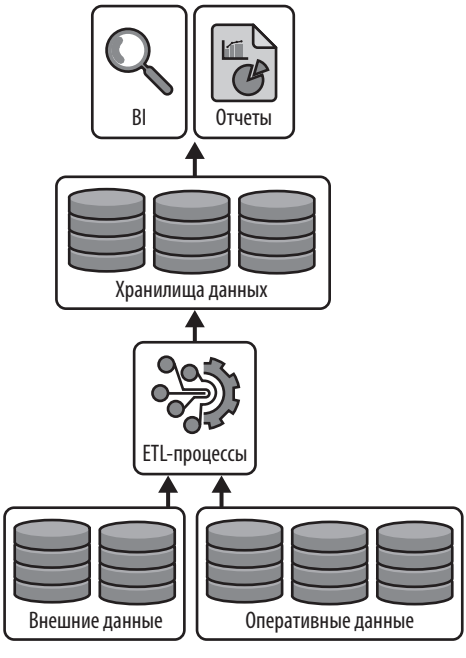

Рис. 1.1. Хранилища данных специально созданы для обработки запросов и агрегирования структурированных данных

Основанные на ACID-транзакциях, хранилища данных включают в себя функции управления (контроль резервного копирования и восстановления, контроль доступа и т. д.) для упрощения работы с базой данных, а также средства оптимизации производительности (индексы, партиционирование и т. д.) для ускорения процесса предоставления достоверных результатов конечному пользователю. Несмотря на свою надежность, хранилища данных зачастую бывает сложно масштабировать под работу с большими объемами данных, разнообразной аналитикой (включая обработку событий и анализ данных) и высокой скоростью обработки данных, характерных для сценариев с big data. Это ограничение является критическим фактором и часто требует обращения к более масштабируемым решениям, таким как озера данных или фреймворки для распределенной обработки данных типа Apache Spark.

Озера данных

Озера данных — это масштабируемые хранилища (HDFS, облачные объектные хранилища, такие как Amazon S3, ADLS Gen2, GCS и т. д.), которые хранят огромные объемы необработанных данных в их исходном формате до тех пор, пока они не понадобятся (рис. 1.2). В отличие от традиционных баз данных, озера данных предназначены для обработки с высокой скоростью данных в масштабах интернета и, кроме того, разнообразных по структуре (например, структурированных, полуструктурированных и неструктурированных данных). Такие характеристики обычно ассоциируются с понятием big data. Озера данных изменили наше понимание о способах хранения и осуществления запросов к большим объемам данных, поскольку они разработаны таким образом, что рабочая нагрузка распределяется между несколькими машинами или узлами. Это файловые системы, которые работают на кластерах недорогого стандартного оборудования. Традиционно хранилища данных масштабировались в рамках одной машины; следует отметить,

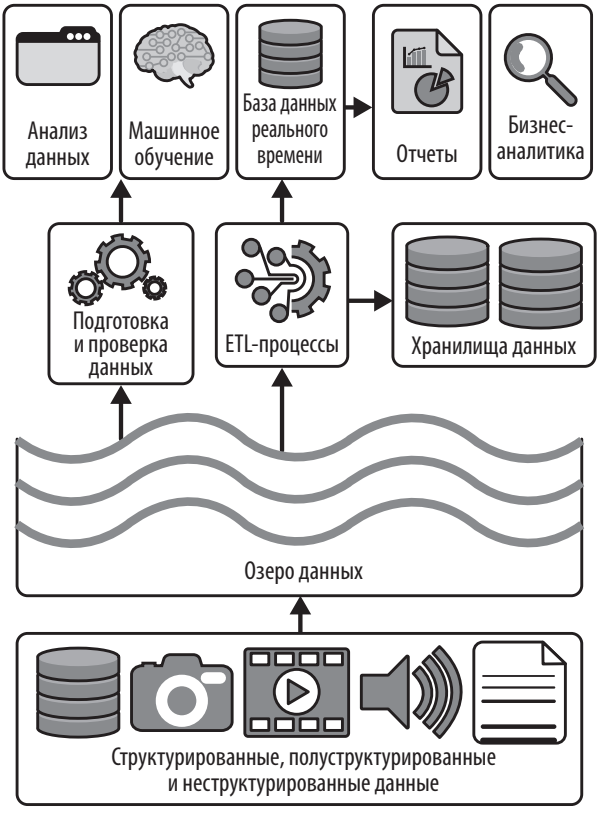

Рис. 1.2. Озера данных предназначены для хранения структурированных, полуструктурированных и неструктурированных данных в масштабируемой инфраструктуре хранения (например, HDFS или облачных объектных хранилищах)

что хранилища данных с массово-параллельной обработкой существуют уже довольно давно, но они дороги и сложны в обслуживании. Кроме того, в то время как хранилища данных были разработаны для структурированных (или табличных) данных, озера данных могут хранить данные в любом выбранном формате, что дает разработчикам гибкость в их хранении.

Хотя озера данных способны работать с любыми вашими данными для задач data science и машинного обучения, они, по сути, являются не совсем надежной формой хранения. Вместо обеспечения защиты согласно принципам ACID эти системы следуют модели BASE (Basically Available, Soft state, Eventual consistency) — базовая доступность, мягкое состояние и конечная согласованность (https://oreil.ly/eAqrW). Отсутствие ACID-гарантий означает, что сбои в работе системы хранения данных приводят к несогласованному состоянию хранилища, где присутствуют файлы-сироты (orphaned files). Последующие запросы к системе хранения включают файлы, которые не должны приводить к дублированию данных (то есть к неправильным ответам).

В совокупности эти недостатки могут привести к созданию инфраструктуры, плохо подходящей для BI-запросов, нестабильной и медленной работе, а также к довольно сложной настройке. Зачастую создание озер данных приводит к появлению ненадежных «болот данных» вместо чистых хранилищ данных из-за отсутствия защиты транзакций, управления схемами и пр.

Озера-хранилища (или озера-хранилища данных)

Озеро-хранилище сочетает в себе лучшие элементы озер данных и хранилищ данных для OLAP-задач. Оно совмещает масштабируемость и гибкость озер данных с функциями управления и оптимизацией производительности хранилищ данных (рис. 1.3). Ранее уже предпринимались попытки обеспечить параллельное сосуществование хранилищ данных и озер данных. Однако такой подход оказался слишком дорогостоящим, усложнял управление, приводил к дублированию



Рис. 1.3. Озера-хранилища — это лучшее из двух миров: хранилищ данных и озер данных

данных и необходимости согласования отчетности, аналитики и анализа данных между отдельными системами. По мере развития практики инженерии данных родилась концепция *озера-хранилища*. Озеро-хранилище устраняет необходимость в разрозненных системах и предоставляет единую согласованную платформу для всех форм анализа данных. Озера-хранилища повышают производительность запросов к данным и упрощают управление ими, облегчая организациям извлечение полезной аналитической информации из своих данных.

Delta Lake, Apache Iceberg и Apache Hudi — самые популярные форматы с открытым исходным кодом для озер-хранилищ. Как нетрудно догадаться, в этой книге основное внимание будет уделено Delta Lake[1].

От проекта Tahoe к Delta Lake: первые годы месяцы

Онлайн-семинар 2021 года «От Tahoe к Delta Lake» (From Tahoe to Delta Lake) (https://oreil.ly/52sIg) позволил с ностальгией оглянуться назад и посмотреть, как создавался Delta Lake. В дискуссии приняли участие разработчики «старой школы» и сопровождающие проекта Delta Lake Бурак Явуз (Burak Yavuz), Дэнни Ли (Denny Lee), Райан Чжоу (Ryan Zhu) и Татхагата Дас (Tathagata Das), а также создатель Delta Lake Майкл Армбруст (Michael Armbrust). В ней также участвовали разработчики Delta Lake «новой школы», создавшие проект delta-rs, Цинпин Хоу (QP Hou) и Р. Тайлер Крой (R. Tyler Croy).

Первоначальное название проекта Delta Lake было «Project Tahoe» (Проект Tahoe), поскольку идея обеспечить надежность транзакций для озер данных впервые пришла в голову Майклу Армбрусту во время катания на лыжах на озере Тахо в 2017 году. Озеро Тахо — это знаковое и огромное озеро в Калифорнии, символизирующее масштабное озеро данных, которое предполагалось создать в рамках данного проекта. Майкл является одним из разработчиков/членов комитета управления проектом Apache Spark™; разработчиком Delta Lake; одним из первых создателей Spark SQL, Structured Streaming и Delta Lake; а также выдающимся инженером-программистом в Databricks. Переход от названия «Tahoe» к «Delta Lake» произошел накануне нового 2018 года и был инициирован Джулсом Дамджи (Jules Damji). Обоснованием смены названия стало желание вызвать в воображении природный процесс, когда реки формируют в устье дельту, образуя отложения наносов, которые в конечном итоге накапливаются и создают плодородную почву для сельскохозяйственных культур. Эта метафора идеально подходила для проекта, поскольку она олицетворяла слияние потоков данных в управляемое озеро данных, где специалисты-практики по работе с данными

[1] Чтобы узнать больше об озерах-хранилищах данных, см. документацию конференции CIDR 2021 (Conference on Innovative Data Systems Research 2021) «Lakehouse: A New Generation of Open Platforms That Unify Data Warehousing and Advanced Analytics» («Озеро-хранилище данных: новое поколение открытых платформ, объединяющих хранилища данных и расширенную аналитику») (https://oreil.ly/ObS0C).

могли бы получать ценную аналитическую информацию. Название Delta также перекликалось с архитектурой проекта, которая была разработана для обработки огромных объемов и высокоскоростных потоков данных, позволяя обрабатывать данные и разделять их на различные потоки или представления.

Но для чего Армбруст создал Delta Lake? Он спроектировал его, чтобы устранить ограничения синхронизации файлов в Apache Spark. В частности, он хотел выполнять операции с большими объемами данных, и ему требовалась надежная поддержка транзакций. Таким образом, его мотивацией к разработке Delta Lake была потребность в масштабируемом журнале транзакций, способном обрабатывать огромные объемы данных и производить сложные операции.

На раннем этапе создания формата Delta Lake было два примечательных случая, которые подчеркнули его эффективность и масштабируемость. Корпорация Comcast начала применять технологию Delta Lake в качестве усовершенствования своих платформ аналитики данных и машинного обучения, а также для управления петабайтами данных. Такой переход позволил сократить эксплуатацию вычислительных ресурсов с 640 до 64 виртуальных машин и уменьшить затраты на обслуживание с 84 до трех рабочих мест. Оптимизировав обработку данных с помощью Delta Lake, компания Comcast сократила мощности вычислительных ресурсов в 10 раз, а количество рабочих мест — в 28 раз. А отдел информационной безопасности Apple использовал Delta Lake для обнаружения угроз и реагирования на них в режиме реального времени, обрабатывая более 300 миллиардов событий в день и ежедневно записывая сотни терабайт данных. Оба случая иллюстрируют превосходную производительность и экономическую эффективность Delta Lake по сравнению с традиционными методами управления данными. Другие примеры мы дополнительно рассмотрим в главе 11.

Что такое Delta Lake?

Delta Lake — это слой хранения данных с открытым исходным кодом, который обеспечивает поддержку ACID-транзакций и масштабируемую обработку метаданных, а также унификацию потоковой и пакетной обработки данных. Изначально он был задуман для работы с Apache Spark и с большими объемами данных в озерах данных.

С помощью Delta Lake вы в состоянии создать единую платформу данных с высокопроизводительной системой обработки запросов по вашему выбору для решения широкого спектра задач, включая бизнес-аналитику (BI), потоковую аналитику/обработку сложных событий, анализ данных и машинное обучение (но не ограничиваясь ими), как показано на рис. 1.4.

Однако по мере своего развития технология Delta Lake была оптимизирована для эффективной работы с самыми разными типами задач (малыми, средними и большими объемами данных и т. д.). Delta Lake также поддерживает работу с различными фреймворками (Apache Spark, Apache Flink, Trino, Presto, Apache

Hive и Apache Druid), сервисами (Athena, Big Query, Databricks, EMR, Fabric, Glue, Starburst и Snowflake) и языками программирования (.NET, Java, Python, Rust, Scala, SQL и т. д.).

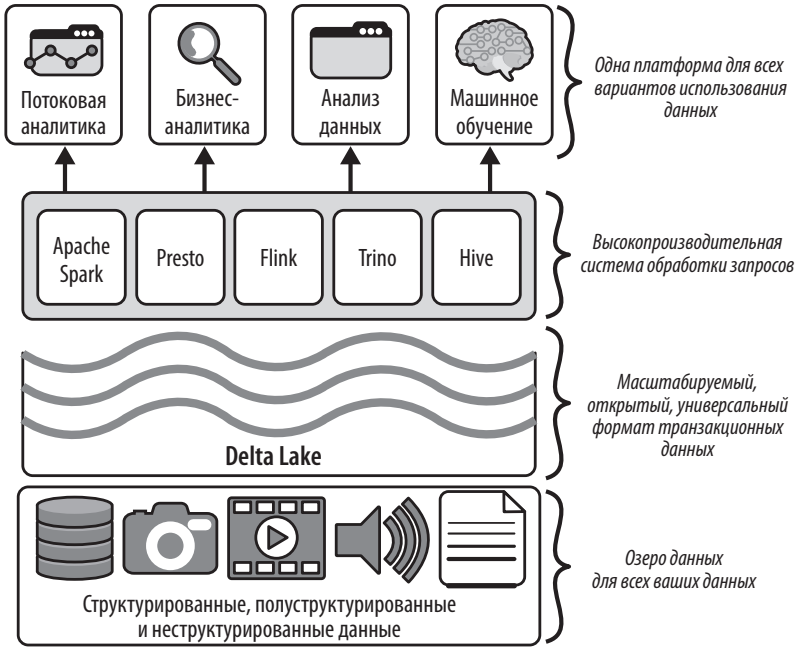

Рис. 1.4. Delta Lake предоставляет масштабируемый, открытый, универсальный формат транзакционных данных для вашего озера данных

Распространенные варианты применения

Разработчики всех типов, от стартапов до крупных предприятий, обращаются к Delta Lake для управления большими данными и решения задач, связанных с искусственным интеллектом. К наиболее распространенным вариантам относятся:

Модернизация озер данных

Delta Lake помогает организациям модернизировать свои озера данных, предоставляя ACID-транзакции, масштабируемую обработку метаданных и принудительное применение схемы, тем самым обеспечивая надежность данных и повышение производительности.

Хранение данных

Существуют как технологии, так и *методы* хранения данных. Формат Delta Lake для архитектуры Lakehouse обеспечивает применение разных методов для поддержания высокой производительности запросов для различных аналитических задач, гарантируя при этом надежность данных.

Машинное обучение / анализ данных (data science)

Delta Lake предоставляет надежную основу для доступа к данным и их обработке, гарантируя командам, занимающимся машинным обучением и data science, быстрое создание и развертывание моделей.

Потоковая обработка данных

Delta Lake сочетает в себе потоковую и пакетную обработку данных. Это позволяет программистам обрабатывать данные в режиме реального времени и выполнять сложные преобразования оперативно по ходу деятельности.

Инженерия данных

Delta Lake предоставляет надежную и высокопроизводительную платформу для команд инженеров данных, дающую возможность создавать конвейеры данных и управлять ими, обеспечивая при этом качество и точность данных.

Бизнес-аналитика

Delta Lake поддерживает SQL-запросы, что упрощает бизнес-пользователям доступ к данным и проведение их анализа, способствуя принятию бизнес-решений на основе данных.

В целом Delta Lake внедряется различными командами, включая инженеров данных, специалистов в области data science и бизнес-пользователей, для управления и анализа больших данных и задач, связанных с применением искусственного интеллекта, обеспечивая надежность данных, производительность и масштабируемость.

Основные функции

Delta Lake включает в себя следующие базовые функции, которые являются основополагающими для открытого формата озер-хранилищ данных (более подробное описание этих функций см. в исследовательской статье, опубликованной в журнале Proceedings of the VLDB (Very Large Data Base) Endowment, под названием «Delta Lake: высокопроизводительное хранилище ACID-таблиц в облачных объектных хранилищах» (Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores) (https://oreil.ly/PaRNO):

ACID-транзакции

Delta Lake гарантирует, что изменения данных выполняются атомарно, последовательно, изолированно и надежно, то есть согласно принципам ACID (атомарность, согласованность, изоляция, долговечность) защиты транзакций. Это означает, что при одновременном (конкурирующем) обращении к данным нескольких клиентских приложений или задач система сохраняет целостность данных. Например, если во время изменения данных происходит сбой процесса, Delta Lake откатывает обратно все изменения, обеспечивая согласованность данных.

Масштабируемые метаданные

Метаданные таблицы в Delta Lake представляют собой журнал транзакций, который обеспечивает транзакционную согласованность в соответствии с вышеупомянутыми принципами ACID транзакций. При оперировании таблицами петабайтного размера поддержка метаданных таблицы бывает чрезвычайно сложной. Механизм масштабируемой обработки метаданных в Delta Lake предназначен для эффективного управления метаданными в больших наборах данных без ущерба для производительности запросов или обработки.

«Путешествие во времени»

Механизм так называемого путешествия во времени в Delta Lake позволяет запрашивать предыдущие версии таблицы для доступа к ретроспективным данным. Благодаря журналу транзакций Delta, этот механизм позволяет указывать версию или временную метку (timestamp), чтобы запросить конкретную версию данных. Это очень полезно при некоторых практических вариантах, таких как аудит данных, соблюдение нормативных требований и восстановление данных.

Единая пакетная / потоковая обработка

Delta Lake был разработан совместно с Apache Spark Structured Streaming с целью упрощения логики для потоковой обработки. Вместо отдельных API для пакетной и потоковой обработки Structured Streaming использует один и тот же API Datasets/DataFrame, работающий в памяти, для обоих сценариев. Это дает возможность разработчикам задействовать одну и ту же бизнес-логику и API, единственным различием является задержка. Delta Lake обеспечивает выполнение принципов ACID в системе хранения для поддержки такого рода унификации.

Эволюция схемы / принудительное применение схемы

Эволюция и принудительное внедрение схемы в Delta Lake обеспечивают согласованность и качество данных, принудительно применяя схему к операциям записи, при этом давая возможность пользователям изменять схему, не нарушая существующие запросы. Эти функции также не дают разработчикам случайно вставить данные с неправильными столбцами или некорректными типами, что, безусловно, важно для поддержания качества и согласованности данных.

История аудитов (журнал аудита)

Эта функция предоставляет подробные журналы всех изменений, внесенных в данные, включая информацию о том, кто внес каждое изменение, что именно было изменено и когда это было сделано. Это крайне важно в плане соблюдения нормативных требований, поскольку пользователи могут отслеживать изменения данных с течением времени и получают гарантию корректности внесения изменений. Все это становится возможным благодаря журналу транзакций Delta.

Операции DML

Delta Lake был одним из первых форматов для озер-хранилищ данных, который предоставил возможность применения операторов языка манипулирования данными (DML, Data Manipulation Language). Изначально это расширяло возможности Apache Spark для поддержки различных операций, таких как вставка, обновление, удаление и слияние (или CRUD-операций). Сегодня пользователи могут эффективно изменять данные, прибегая к различным фреймворкам, сервисам и языкам.

Открытый исходный код

Delta Lake был создан в рамках компании Databricks, обладающей обширным опытом в области разработки ПО с открытым исходным кодом (основатели Databricks были первыми создателями Apache Spark). Вскоре после своего появления Delta Lake был передан в дар некоммерческому консорциуму Linux Foundation, чтобы разработчики могли свободно внедрять, изменять и распространять это программное обеспечение, при этом способствуя сотрудничеству и продвижению инноваций в сообществе специалистов по инженерии данных.

Производительность

Несмотря на то что Delta Lake является форматом хранения данных для озер хранилищ, он разработан оптимально с точки зрения повышения скорости запросов и обработки данных как при их загрузке, так и при выполнении запросов в рамках конфигурации по умолчанию. Хотя производительность Delta Lake предрасположена к постоянной подстройке, в большинстве случаев настройки по умолчанию будут хорошо работать для всех ваших сценариев.

Простота использования

Delta Lake изначально разрабатывался с упором на простоту. Например, чтобы записать таблицу с помощью Apache Spark в файл формата Parquet, необходимо выполнить следующую команду:

```
data.write.format("parquet").save("/tmp/parquet-table")
```

Чтобы сделать то же самое для формата Delta, выполните команду:

```
data.write.format("delta").save("/tmp/delta-table")
```

Структура таблиц Delta Lake

Таблица Delta Lake, или просто таблица Delta, состоит из нескольких ключевых компонентов, которые, работая совместно, предоставляют пользователю надежное, масштабируемое и эффективное решение для хранения данных. Перечислим основные элементы.

Файлы данных

В таблицах Delta Lake данные хранятся в файлах формата Parquet. Эти файлы содержат фактические данные и находятся в распределенной облачной или локальной файловой системе хранения, такой как HDFS (Hadoop Distributed File System), Amazon S3, Azure Blob Storage (или Azure Data Lake Storage [ADLS] Gen2), GCS (Google Cloud Storage) или MinIO. Формат Parquet был выбран за его эффективность при хранении больших наборов данных и выполнении запросов к ним.

Журнал транзакций

Журнал транзакций, также известный как журнал Delta, является важным компонентом Delta Lake. Он представляет собой упорядоченную запись всех транзакций, выполненных с таблицей Delta Lake. Журнал транзакций обеспечивает соответствие принципам ACID, записывая все внесенные в таблицу изменения в виде последовательности JSON-файлов. Каждая транзакция записывается в каталог `_delta_log` в новый JSON-файл, который содержит метаданные о транзакции, такие как выполненная операция, добавленные или удаленные файлы и схема таблицы на момент транзакции.

Метаданные

Метаданные в Delta Lake включают в себя информацию о схеме таблицы, партиционировании и параметрах конфигурации. Эти метаданные хранятся в журнале транзакций и могут быть извлечены с помощью API для работы с SQL, Spark, Rust и Python. Метаданные помогают управлять таблицей и оптимизировать ее, предоставляя информацию для принудительного применения схемы (schema enforcement) и эволюции схемы (schema evolution), стратегий партиционирования (partitioning strategies) и пропуска данных (data skipping).

Схема

Схема таблицы Delta Lake определяет структуру данных, включая столбцы, типы данных и т. д. Схема применяется принудительно при записи, обеспечивая соответствие всех данных, записываемых в таблицу, определенной заданной структуре. Delta Lake поддерживает эволюцию схемы (добавление новых столбцов, переименование столбцов и пр.), что позволяет обновлять схему с течением времени по мере изменения данных.

Контрольные точки

Контрольные точки (Checkpoints) — это периодически создаваемые мгновенные снимки состояния (снапшоты) журнала транзакций, которые помогают ускорить процесс восстановления. Delta Lake по умолчанию консолидирует состояние журнала транзакций каждые 10 транзакций. Это позволяет клиентским процессам чтения быстро восстановить актуальное состояние, начиная с самой свежей контрольной точки, а не воспроизводить весь журнал

транзакций с самого начала. Контрольные точки хранятся в виде файлов Parquet и автоматически создаются Delta Lake.

На рис. 1.5 схематически представлена структура таблицы Delta Lake.

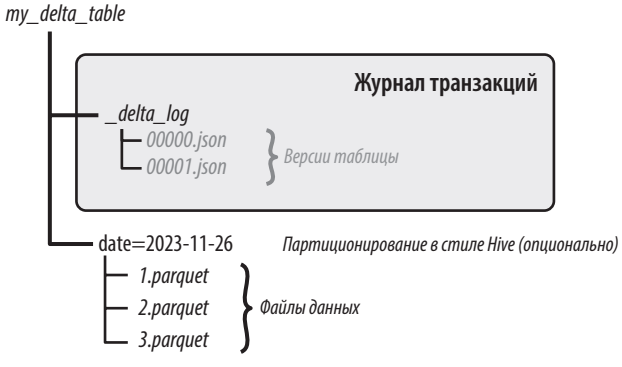

Рис. 1.5. Структура таблицы Delta Lake, включая журнал транзакций и файлы данных (изображение взято из блога Дэнни Ли)[1]

Протокол транзакций Delta

В предыдущем разделе была описана структура таблиц Delta Lake. Протокол журнала транзакций Delta (https://oreil.ly/CPxmm) — это спецификация, определяющая, как клиентские приложения должны взаимодействовать с таблицей согласованным образом. По сути, любое взаимодействие с таблицей Delta должно начинаться с чтения журнала транзакций Delta, чтобы определить, какие файлы следует читать. Когда клиентское приложение изменяет данные, оно инициирует создание новых файлов данных (а именно файлов Parquet), а затем вставляет новые метаданные в журнал транзакций для фиксации изменений в таблице. Фактически многие из первоначальных интеграций Delta Lake (https://oreil.ly/vfuS2) (Delta-Spark, коннектор Trino, Delta-Rust API и т. д.) имели кодовые базы, поддерживаемые различными сообществами разработчиков. Клиентское приложение на Rust могло записывать данные, Spark-клиент мог их изменять, а клиент Trino мог одновременно с ними читать данные из одной и той же таблицы Delta без возникновения конфликтов, поскольку все они независимо друг от друга следовали одному и тому же протоколу.

Реализация этой спецификации привносит ACID-свойства в большие коллекции данных, хранящихся в виде файлов в распределенной файловой системе или объектном хранилище. Как определено в спецификации, протокол был разработан с учетом определенных целей.

---

[1] Дэнни Ли (Denny Lee), «Разбираемся с журналом транзакций Delta Lake на уровне файлов» (Understanding the Delta Lake Transaction Log at the File Level) (https://oreil.ly/X5p5R), Denny Lee (блог), 26 ноября 2023 г.

Сериализуемость ACID-записей

Несколько процессов записи могут параллельно изменять таблицу Delta, поддерживая при этом свойства ACID.

Изоляция снапшотов для чтения

Пользователи в состоянии прочитать согласованный снапшот таблицы Delta даже при наличии параллельных записей.

Масштабируемость до миллиардов разделов или файлов

Запросы к таблице Delta могут планироваться на одном компьютере или параллельно.

Самоописание

Все метаданные для таблицы Delta хранятся вместе с данными. Такая архитектура устраняет необходимость в поддержке отдельного хранилища метаданных для чтения данных и дает возможность копировать или перемещать статичные таблицы с помощью стандартных инструментов файловой системы.

Поддержка инкрементальной обработки

Пользователи отслеживают журнал Delta, чтобы определить, какие данные были добавлены за определенный период времени, что позволяет обеспечить эффективную потоковую обработку.

Знакомство с журналом транзакций Delta Lake на уровне файлов

Чтобы лучше понять, как это работает, давайте посмотрим, что происходит на уровне файлов при создании таблицы Delta. Изначально журнал транзакций таблицы автоматически создается в подкаталоге `_delta_log`. По мере внесения изменений в таблицу операции записываются в журнале транзакций в виде упорядоченных атомарных коммитов. Каждый коммит записывается в виде JSON-файла, начиная с `000...00000.json`. Дальнейшие изменения в таблице генерируют последующие JSON-файлы в порядке возрастания номеров, так что следующие коммиты записываются как `000...00001.json`, `000...00002.json` и т. д. Каждое приращение номера JSON-файла представляет собой новую версию таблицы, как показано на рис. 1.5.

Обратите внимание, что структура файлов данных не изменилась; они существуют в виде отдельных файлов Parquet, сгенерированных движком запросов или языковой средой, выполняющей запись в таблицу Delta. Если ваша таблица использует партиционирование по типу Hive, структура сохранится.

Единый источник истины

Delta Lake позволяет нескольким операциям чтения и записи данных работать с таблицей одновременно. Это центральный репозиторий, который отслеживает

все изменения, внесенные пользователями в таблицу. Эта концепция важна, поскольку со временем задания по обработке данных в вашем озере данных неизбежно будут завершаться сбоем. В результате останутся неудаленные частичные файлы (partial files). Последующая обработка или запросы не смогут определить, какие файлы следует включать в запросы, а какие нет. Для того чтобы пользователям в каждый момент времени демонстрировалось корректное состояние данных, журнал транзакций Delta сохраняется как *единый источник истины*.

Связь между метаданными и данными

Поскольку журнал транзакций Delta является единым источником истины, любой клиент, которому необходимо выполнить чтение или запись данных в вашу таблицу Delta, должен сначала запросить данные из журнала транзакций. Например, при вставке данных при создании нашей таблицы Delta мы сначала создаем два файла Parquet: `1.parquet` и `2.parquet`. Это событие автоматически добавляется в журнал транзакций и сохраняется на диске как коммит `000...00000.json` (см. А на рис. 1.6).

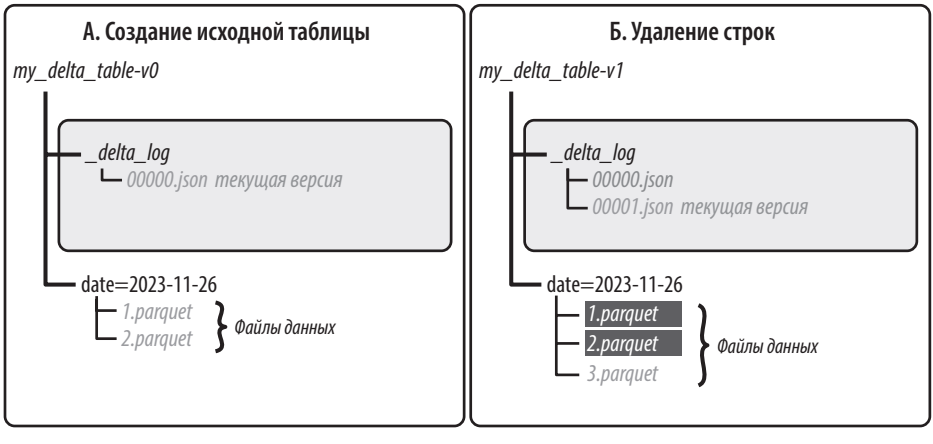

Рис. 1.6. *Слева*: создание новой таблицы Delta путем добавления файлов Parquet и их взаимосвязь с журналом транзакций Delta. *Справа*: удаление строк из этой таблицы Delta путем удаления и добавления файлов и их взаимосвязь с журналом транзакций Delta

В следующей команде (Б на рис. 1.6) мы выполняем операцию `DELETE`, которая приводит к удалению строк из таблицы. Вместо изменения существующих файлов Parquet (`1.parquet`, `2.parquet`) Delta создает третий файл (`3.parquet`).

Рассмотрение файлов и данных в рамках многоверсионного управления конкурентностью (MVCC)

Для удаления данных в объектных хранилищах быстрее создать новый файл или файлы, содержащие незатронутые строки, чем изменять существующие файлы

Parquet. Кроме того, такой подход предоставляет все преимущества многоверсионного управления конкурентным доступом (MVCC). Multiversion Concurrency Control — это метод оптимизации базы данных, при котором создаются копии данных, что позволяет одновременно безопасно читать и обновлять данные. Этот метод также позволяет Delta Lake обеспечивать механизм «путешествия во времени». Таким образом, Delta Lake создает несколько файлов при осуществлении этих действий, обеспечивая атомарность, MVCC и скорость.



Можно ускорить этот процесс, используя векторы удаления, — подход, который будет описан в главе 8.

Удаление/создание файлов Parquet, показанное на панели Б рис. 1.6, происходит в рамках одной транзакции, записанной в журнале транзакций Delta в файле `000...00001.json`.

Некоторые важные замечания относительно атомарности:

- Если пользователь будет читать файлы Parquet, не заглядывая в журнал транзакций Delta, он будет читать дубликаты из-за реплицированных строк во всех файлах (`1.parquet`, `2.parquet`, `3.parquet`).
- Все действия по удалению и добавлению данных хранятся в едином журнале транзакций `000...00001.json`. В тот момент, когда клиентское приложение запрашивает таблицу Delta, система записывает оба этих действия и пути к файлам (filepaths) для соответствующего снапшота (snapshot). Для такой транзакции будет прописан только путь к файлу `3.parquet`.
- Обратите внимание, что операция удаления представляет собой «мягкое удаление» (soft delete) или создание маркера удаления (tombstone), при котором физическое удаление файлов (`1.parquet`, `2.parquet`) еще не произошло. Физическое удаление файлов произойдет при выполнении команды `VACUUM`.
- В предыдущей транзакции `000...00000.json` путь к файлу указывает на исходные файлы (`1.parquet`, `2.parquet`). Таким образом, при запросе более старой версии таблицы Delta с помощью механизма «путешествия во времени» журнал транзакций указывает на файлы, составляющие этот более старый снапшот.

Обзор взаимодействия между метаданными и данными

Теперь мы уже лучше понимаем, что происходит на уровне *отдельных* файлов данных и метаданных, но как все это работает вместе? Давайте рассмотрим эту задачу, следуя схеме на рис. 1.7, которая представляет типичный сценарий сбоя обработки данных. Таблица изначально представлена двумя файлами Parquet (`1.parquet` и `2.parquet`) в момент времени t_0.