ГЛАВА 1

Эволюция архитектуры данных

Создание надежной архитектуры данных — одна из самых сложных задач при работе с данными. Процесс обработки данных — от их сбора до преобразования, распространения и итогового потребления — сильно зависит от множества разных факторов: средств управления данными (governance), используемых инструментов, профиля рисков организации, размера, зрелости, требований сценариев использования и других потребностей, включая производительность, гибкость и управление затратами.

Несмотря на эти отличия, каждая архитектура данных включает несколько основных компонентов. Я часто рассматриваю их через призму трехуровневой архитектуры — концепцию, которую я представил в своей предыдущей книге «Data Management at Scale»[1] (O'Reilly). Этот дизайн стал полезным инструментом концептуализации и структурирования стратегий управления данными предприятий. Он включает три уровня: на первом размещаются поставщики данных, второй служит платформой распространения, а третий включает потребителей данных. Кроме того, всеобъемлющий уровень метаданных и управления необходим для контроля за всей архитектурой данных. Высокоуровневая схема этого дизайна изображена на рис. 1.1.

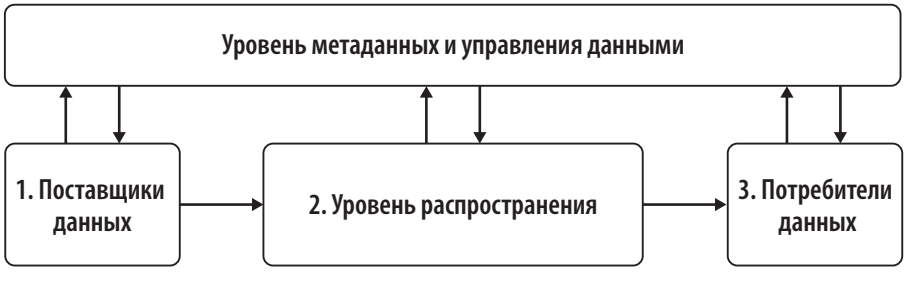

Рис. 1.1. Трехуровневая архитектура

[1] Стренгхольт П. «Масштабируемые данные. Высоконагруженные архитектуры, Data Mesh и Data Fabric», 2-е изд.

Краткое описание каждого уровня (слева направо):

Первый уровень

Уровень включает поставщиков данных, представляющих источники, из которых извлекаются данные. Извлеченные данные характеризуются разными типами, форматами и местоположениями, распределенными по разным организациям.

Второй уровень

Этот уровень представляет платформу распространения. Для него характерна высокая сложность, обусловленная огромным выбором доступных инструментов и технологий. Организации сталкиваются с непростой задачей выбора из сотен (если не тысяч) продуктов и решений с открытым кодом для проведения интеграции.

Третий уровень

Этот уровень включает потребителей данных, а точнее, сервисы-потребители данных. Сервисы данных используют бизнес-аналитику, машинное обучение и искусственный интеллект (ИИ) для построения прогнозов, автоматизации и выявления закономерностей в реальном времени. Другие сервисы обеспечивают основные функции хранения и обработки данных. Этот уровень включает разнообразные технологии и типы приложений, так как каждая бизнес-задача требует специализированного решения; как следствие, оба типа сервисов играют важную роль в современных архитектурах данных.

Чтобы завершить высокоуровневое описание архитектуры, я обычно рисую охватывающий уровень, который называется уровнем метаданных и управления данными. Этот уровень очень важен для общего контроля и управления всей архитектурой данных.

Трехуровневая диаграмма (в которой особенно важное место занимает внутренняя архитектура среднего уровня) отражает эволюцию управления платформой данных в организациях. Она демонстрирует значительный сдвиг от традиционных проприетарных систем хранилищ данных (data warehouses) к более гибким распределенным архитектурам данных с открытым исходным кодом. Этот переход обусловлен появлением фреймворков и инструментов с открытым кодом, объединяемых общим термином «современный стек данных».

Проблема в том, что современный стек данных сам по себе не представляет завершенной платформы данных. Он требует интеграции многих независимых сервисов и инструментов, каждый из которых проектировался под конкретные элементы обработки и управления данными. Каждый сервис или инструмент приносит свой набор стандартов обмена данными, протоколов безопасности и средств управления метаданными. Более того, пересекающиеся функциональности многих сервисов усложняют развертывание и использование. Таким образом, чтобы

эффективно использовать современный стек данных, необходимо тщательно выбрать наиболее подходящие сервисы, а затем тщательно провести интеграцию каждого компонента. Процесс интеграции существенно затрудняет начало работы.

> «Это упущение не какого-то отдельного поставщика; это упущение рынка в целом»[1].
>
> *Бенн Стэнсил*

Поставщики технологий тоже видят эту проблему. Они признают сложность интеграции и управления инфраструктурой, хранения данных и вычислений. В области разработки и стандартизации был достигнут значительный прогресс, особенно в Apache Spark (https://spark.apache.org) и форматах таблиц с открытым исходным кодом, таких как Delta Lake (https://docs.delta.io/latest/index.html). Это привело к созданию полнофункциональных программных платформ, упрощающих обработку данных. Многие инженеры данных предпочитают эти платформы за их инновационные возможности. Вдобавок организации, пользующиеся Spark и Delta Lake, обнаружили, что *архитектура медальона* (определение которой приводится в следующем разделе) особенно удобна, так как она использует все сильные стороны мощной, масштабируемой и эффективной структуры для сквозного управления данными и аналитики.

Что такое архитектура медальона?

Архитектура медальона — это паттерн проектирования данных, предназначенный для их логической организации (чаще всего в озерах-хранилищах). В нем используются три уровня платформы данных, на каждом из которых происходит пошаговое и последовательное улучшение структуры и качества данных (бронзовый ⇒ серебряный ⇒ золотой уровни). В главе 3 каждый из уровней будет рассмотрен более подробно, а пока я ограничусь их кратким описанием:

Бронзовый уровень (Bronze)

На бронзовом уровне хранятся сырые данные из разных источников в их исходной структуре. Уровень служит точкой отсчета изменений и надежным исходным хранилищем.

Серебряный уровень (Silver)

На серебряном уровне данные очищаются и стандартизируются для комплексной аналитики посредством проверки качества, стандартизации, удаления дубликатов и других преобразований. Это переходный этап, на котором получают обработанные детализированные согласованные данные лучшего качества.

---

[1] Цитата из статьи Бенна Стэнсила (Benn Stancil) Microsoft Builds the Bomb (https://oreil.ly/FMlrI), в которой рассматриваются проблемы решений платформ данных на уровне рынка.

Золотой уровень (Gold)

На золотом уровне очищенные данные оптимизируются для конкретных бизнес-идей и решений. На нем происходит агрегирование, обобщение и обогащение данных для получения высокоуровневых отчетов и аналитики, при этом на первый план выходят производительность и масштабируемость, обеспечивающие быстрый доступ к ключевым метрикам и аналитике.

Такая структура, изображенная на рис. 1.2, отлично подходит для практической реализации или сценариев использования, способствующих развитию бизнеса.

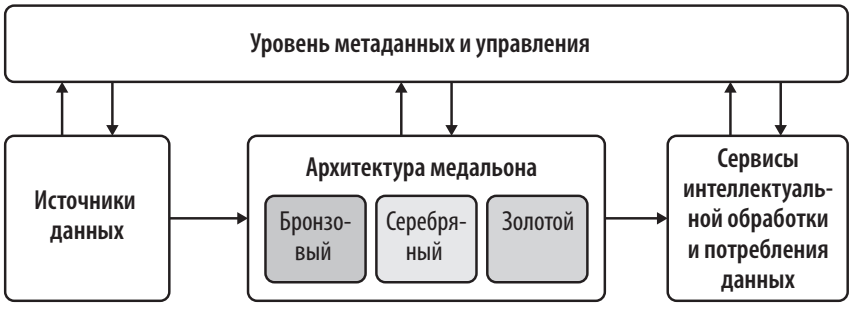

Рис. 1.2. В архитектуре медальона данные упорядочиваются по трем уровням, на каждом из которых структура и качество данных постепенно улучшаются

Архитектура медальона предоставляет обозначения разных уровней, удобные для стороны бизнеса. Тем не менее многие организации не могут понять, как организовать эффективное распределение по уровням и моделирование своих данных, и тратят бесчисленные часы на обсуждение таких вопросов, как выбор, интеграция, пересечение функциональности и т. д. У них возникают трудности с распределением целей по разным зонам или пониманием смысла уровней Bronze, Silver и Gold. Также возникают вопросы относительно управления данными и стратегий масштабирования. Например, какие части архитектуры могут управляться метаданными с гибкой настройкой конфигурации и автоматизацией? Выбор полнофункциональной платформы не даст готовых ответов на вопросы и не решит этих проблем.

Прежде чем погружаться в конкретику ответов на эти вопросы и проектирование архитектуры медальона, необходимо понять ход эволюции архитектур данных. Где появились эти платформы? Какие принципы проектирования остаются актуальными и должны применяться и в наши дни? Понимание истории и фундаментальных принципов архитектур данных поможет эффективно использовать эти сквозные платформы. Эта глава как раз и содержит такую вводную информацию. В ней исследуются прошлые достижения, наблюдения, паттерны, лучшие практики и принципы. Вооружившись необходимой информацией, обоснованиями и выводами, вы будете готовы к части II, в которой научитесь проектировать и реализовывать собственные архитектуры данных.

Если вы чувствуете, что уже достаточно хорошо разбираетесь в основах архитектуры данных, можете перейти к главе 3, в которой дается подробное описание архитектуры медальона и ее уровней. Если нет — начинайте вместе с нами знакомиться с традиционными хранилищами данных. Затем мы перейдем к исследованию паттернов, появившихся с озерами данных (data lakes), включая Hadoop. Мы обсудим достоинства, недостатки и выводы, которые были сделаны из каждой архитектуры, а также связь этих разработок с современными лучшими практиками. Наконец, мы подробно рассмотрим озера-хранилища и архитектуры медальона, между которыми существует тесная связь. В этом разделе мы также обсудим разных поставщиков технологий.

Краткая история архитектуры хранилищ данных

Мысленно вернемся в 1990-е годы. Тогда для сбора и интеграции данных в однородные коллекции стали использоваться хранилища данных (data warehouse). Они позволяли создать единую версию истинных данных, служащую ключевым источником информации для принятия бизнес-решений в компании.

Процесс получения аналитики на основе данных состоит из нескольких этапов, включая сбор данных из разных источников, приведение их к единому формату и загрузку в центральный репозиторий. Этот процесс более подробно рассматривается в главе 3. А пока сосредоточимся на архитектуре хранилища данных (рис. 1.3), которая также включает сервисы-источники и сервисы-потребители (например, средства составления отчетов).

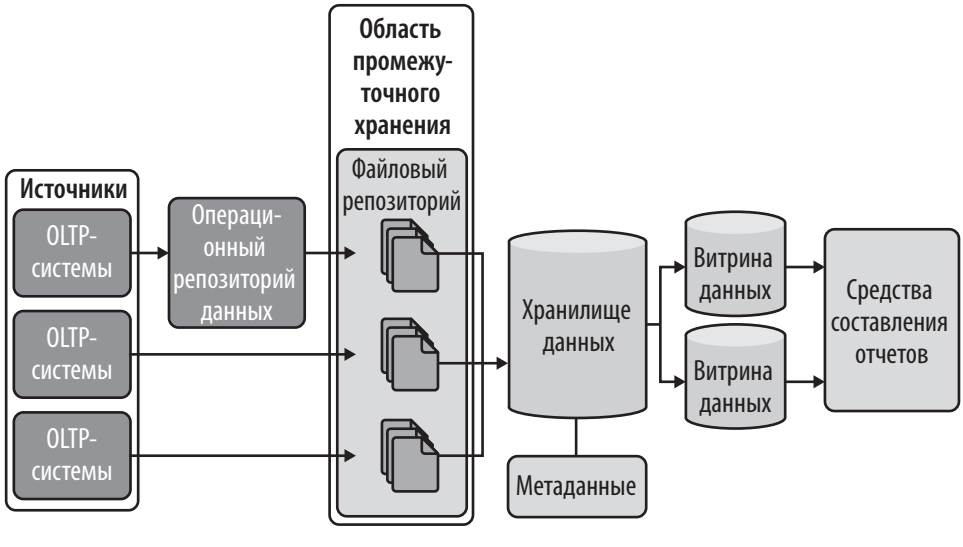

Рис. 1.3. Типичная архитектура хранилища данных

Начнем с рассмотрения уровней от OLTP-систем (слева) до хранилищ данных в середине. Витрины данных (data marts) в правой части рассматриваются в разделе «Методология Инмона», с. 33.

OLTP-системы

Бóльшая часть систем-источников проектировалась для транзакционных или эксплуатационных целей, что отражало потребности в управлении транзакциями и хранении записей в эпоху ранних вычислительных технологий. Такие источники, показанные на рис. 1.3 слева, часто назывались системами онлайн-обработки транзакций, или *OLTP-системами* (OnLine Transaction Processing), что описывало их основное назначение.

Внимательнее присмотревшись к OLTP-системе, можно заметить, что рабочие нагрузки в ней обычно весьма предсказуемы. Вы должны понимать, как используются OLTP-системы и какие типичные нагрузки можно ожидать. Запросы относительно просты, а объем полученных данных невелик: прочесть запись, обновить, удалить и т. д. Нижележащая физическая модель данных проектируется (оптимизируется) для таких предсказуемых запросов. В результате OLTP-системы обычно нормализуются так, чтобы каждый атрибут хранился только в одном экземпляре.

Нормализованные и денормализованные базы данных

В контексте реляционных баз данных *нормализацией* называется процесс изменения структуры данных для удаления избыточности и улучшения целостности данных. Обычно нормализация представляет собой приведение данных к одной из *нормальных форм*, каждая из которых устраняет конкретные аномалии или избыточность. На практике чаще всего применяется третья нормальная форма (3NF). Нормализация позволяет хранить данные более эффективно и последовательно, упрощая обслуживание и выборку данных. Таким образом, говоря о нормализованных данных, мы имеем в виду данные, организованные с расчетом на эффективность хранения и целостность.

Денормализацией называется процесс отмены эффекта нормализации с намеренным введением в базу данных избыточности с целью ускорения обработки запросов и получения данных. Денормализация часто используется в хранилищах данных и аналитике для оптимизации выборки и обработки данных. Выполняя денормализацию данных, можно сократить количество соединений, необходимых для загрузки данных, что может значительно улучшить производительность запросов. С другой стороны, денормализация также может создавать проблемы целостности данных, так как неправильное управление может привести к несоответствиям в избыточных данных.

Многие OLTP-системы нацелены на обеспечение в первую очередь целостности и стабильности. Для этого они используют системы управления базами данных, поддерживающие свойства ACID: атомарность (Atomicity), согласованность (Consistency), изоляцию (Isolation) и устойчивость (Durability)[1]. Эти свойства важны для эффективного проведения бизнес-транзакций, так как они помогают управлять данными и защищать их в процессе работы. Однако дизайн OLTP влечет за собой ряд последствий, которые необходимо понимать, особенно при обсуждении в организации.

Во-первых, операционные системы не предназначены для того, чтобы предоставлять полный и консолидированный аналитический обзор всего, что происходит в бизнесе или конкретных предметных областях. Дело в том, что извлечение данных из моделей с высокой степенью нормализации для сложных запросов часто затруднительно из-за высокой нагрузки на OLTP-системы. Чтобы получить нужную информацию, требуются сложные запросы. Такие запросы требуют больших объемов данных и их комбинаций, а это означает, что многие таблицы придется соединять или группировать.

Кроме того, запросы такого типа обычно интенсивно расходуют ресурсы, и при слишком частом их выполнении производительность может быть ограниченна, особенно при работе с большими датасетами[2]. Если такие проблемы приводят к аномальному поведению операционной системы, это может отрицательно сказаться на бизнесе. Следовательно, очень важно тщательно продумать потенциальные последствия нормализованной структуры. Хотя нормализация может эффективно работать для некоторых целей, она не всегда подходит для получения больших аналитических представлений данных.

Во-вторых, из-за жестких требований к высокой целостности, производительности и доступности OLTP-системы часто обходятся дорого. Типичная стратегия оптимизации таких систем включает перемещение неиспользуемых данных и/или проектирование систем для работы только с самыми новыми данными. Такой подход означает, что обновление данных будут происходить моментально без сохранения старых версий записей. В контексте виртуализации данных[3] инженеры иногда считают, что все исторические данные должны храниться в OLTP-системах, а не в хранилище, озере данных или озере-хранилище. Но из-за особенностей дизай-

---

[1] Принципы ACID обеспечивают надежность транзакций данных, делая их неделимыми, согласованными, изолированными и устойчивыми, что крайне важно для поддержания целостности данных, особенно в критически значимых областях (финансовых системах и т. д.).

[2] Приложения (например, веб-сервисы), загружающие большой объем данных для одного наблюдения, не обязательно страдают от подобных проблем.

[3] Виртуализация данных — технология, позволяющая управлять и оперировать данными без необходимости их копирования и экспортирования. Фактически она создает виртуальную прослойку, которая отделяет пользователя от технических подробностей данных: местонахождения, структуры, происхождения и т. д.

на OLTP-систем такое решение часто оказывается непрактичным. В некоторых ситуациях хранение огромных объемов исторических данных тормозит работу систем, что приводит к замедлению обработки транзакций и обновлений. Кроме того, могут возникнуть проблемы с обслуживанием и адаптируемостью.

В-третьих, OLTP-системы изначально проектировались с расчетом на оптимизацию для конкретных бизнес-задач, они были изолированными и независимыми. В разных системах данные хранятся по-разному. Изоляция и разнообразие усложняют получение унифицированного представления любой отдельной системы без значительных усилий по интеграции данных.

Отделяя аналитические нагрузки от операционных систем, организации решают многие из этих проблем. Такое разделение не только сохраняет целостность исторических данных, но и оптимизирует системы для более эффективной аналитической обработки. Кроме того, хранение и обработка данных из разных источников в универсальном формате дают более целостное представление, чем могла бы предложить одна система. Стандартная практика заключается в вынесении этих данных на средний уровень, такой как хранилище данных.

Хранилища данных

Хранилище данных (data warehouse) служит центральным хабом. Оно используется для сбора и структурирования данных из разных систем-источников, их преобразования к последовательному формату для аналитической обработки в реальном времени, или OLAP (OnLine Analytical Processing), включающей сложные специфические вычисления. Анализ в автономном режиме обычно менее критичен для бизнеса, и требования к целостности и доступности таких систем могут быть менее жесткими. Хотя данные в OLTP-системах хранятся и оптимизируются в плане целостности и избыточности, в OLAP оптимизируется аналитическая производительность. Поэтому OLAP, где в основном выполняются повторяющиеся операции чтения при редких операциях записи, обычно оптимизируется для интенсивного чтения данных. Например, данные могут дублироваться для паттернов чтения в различных аналитических сценариях. Таблицы в базах данных OLAP обычно не подвергаются жесткой нормализации, но в ходе предварительной обработки преобразуются в денормализованные структуры: как правило, в большие деструктурированные разреженные копии данных.

В книге «Deciphering Data Architectures» Джеймса Серра (James Serra) (издательство O'Reilly)[1] приводится подробный обзор архитектур данных, включая хранилища, озера и озера-хранилища данных. Это ценный источник информации об эволюции архитектур данных и принципов, лежащих в их основе.

[1] Серра Дж. «Архитектуры данных: современные решения для любых задач».

Чтобы загрузить данные в хранилище данных, необходимо первым делом извлечь их из разных систем-источников. Для этого необходимо понимать исходные данные, а также прочитать и скопировать необходимые данные в область, часто называемую *областью промежуточного хранения* (staging area), для последующей обработки. Область промежуточного хранения, как показано на рис. 1.3, располагается между операционными системами-источниками и областью интеграции данных и представления. Она часто представляет собой как область хранения, так и совокупность процессов, объединяемых общим термином ETL (Extract, Transform, Load — извлечение, преобразование и загрузка).

Область промежуточного хранения

Область, или уровень, промежуточного хранения может быть реализована разными способами, которые не ограничиваются реляционными базами данных и файловыми репозиториями. Реляционные базы данных гибкие, но дорогие. Файловые репозитории дешевы, но ограничены по функциональности. Области промежуточного хранения также обычно используются для хранения исторических копий. Это полезно для повторного воспроизведения сценариев, когда поврежденное хранилище данных необходимо восстановить. Количество старых поставок данных (исторических копий) в разных областях промежуточного хранения разных организаций может различаться. Мне знакомы сценарии, когда все поставки данных, включая исправления, должны были храниться для целей аудита в течение нескольких лет. И наоборот, я видел, как область промежуточного хранения очищалась после успешной обработки или по истечении фиксированного периода времени. В этом контексте очистка производится для снижения затрат на хранение или может быть обусловлена требованиями к управлению данными.

Сложности с извлечением и промежуточным хранением связаны с тем, что разные системы-источники могут использовать разные типы формата данных. Следовательно, реальный процесс загрузки данных будет значительно изменяться в зависимости от типа источника данных. Некоторые системы допускают прямые обращения к базе данных, другие требуют, чтобы загрузка, или прием (ingestion), данных производилась через API. Несмотря на технологические достижения, многие процессы сбора данных продолжают зависеть от извлечения файлов, поскольку этот способ оказался более экономичным и простым в реализации для больших объемов данных.

После извлечения данных в область промежуточного хранения к ним применяются различные преобразования: очистка, обогащение, управление мастер-данными и назначение ключей хранилища. Все эти преобразования представляют собой предварительные шаги, в ходе которых данные из разных источников соединяются, преобразуются и загружаются в области интеграции и представления в хранилище данных. В большинстве случаев требуется переработать сильно нормализованные

и сложные структуры данных. Как вы узнали в разделе OLTP, такие структуры поступают непосредственно из транзакционных систем-источников.

Необходимо понимать, что проблемы с преобразованием данных остаются и в современных архитектурах данных. Избежать дилеммы преобразования данных невозможно. Чтобы данные можно было использовать для аналитической обработки, их необходимо очистить и интегрировать.

Итак, остается вопрос: как моделировать данные в областях интеграции и представления? Рассмотрим две основные методологии: Инмона и Кимбалла.

Методология Инмона

К сожалению, в среде дата-инженеров сохраняется недопонимание относительно того, должны ли данные после извлечения и преобразования и перед загрузкой в область представления для генерирования запросов и отчетов моделироваться в физические нормализованные структуры. Это недопонимание обусловлено различием подходов к обработке данных.

Традиционно хранилища данных стоили дорого. Методология Инмона, появившаяся в начале 1990-х и получившая название по фамилии своего создателя Билла Инмона (Bill Inmon), показана на рис. 1.4. Это был популярный метод, основанный на нормализованной модели данных, обычно в третьей нормальной форме.

Модель 3NF структурирует данные в таблицы с минимальной избыточностью; она гарантирует, что каждый блок данных хранится только в одном экземпляре, а дубликаты данных исключены. Она также обеспечивает целостность ссылок, так как каждый непервичный атрибут таблицы зависит только от первичного ключа. Этот метод значительно снижает требования к необходимому пространству хранения. Кроме того, он подразумевает создание централизованного высокоструктурированного хранилища данных, называемого *корпоративным хранилищем данных* (EDW, Enterprise Data Warehouse), которое обслуживает всю организацию.

Для запросов и улучшения производительности методология Инмона также включает уровень представления: *витрины данных* (data marts). Они создаются после надежного сохранения данных на уровне интеграции. Обычно витрины данных содержат лишь небольшое подмножество данных уровня интеграции и проектируются для конкретного сценария использования, группы или круга пользователей. Данные в таких витринах обычно организуются по *схеме «звезда»*, так как они оптимизированы для быстрого чтения. Простота и денормализация структур данных в схемах «звезда» — основные причины того, почему эти схемы хорошо подходят для операций с интенсивным чтением. Соответственно, можно заметить, что данные в витринах данных хранятся менее эффективно по сравнению с уровнем интеграции. Кроме того, преобразование данных из 3NF на уровне интеграции в денормализованную модель в витринах данных потребует значительных усилий. Этот процесс часто включает сложные соединения (joins)

для пересборки данных и полного восстановления их значений для более эффективного анализа и запросов информации[1].

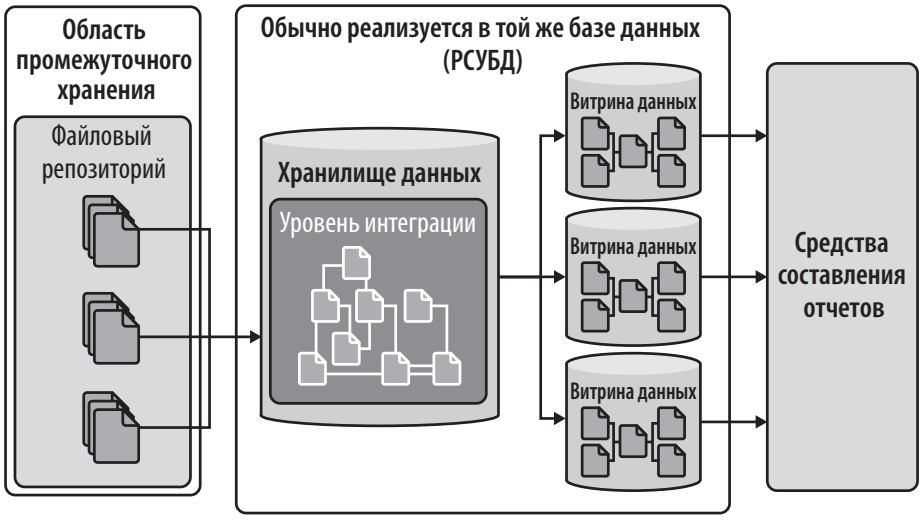

Рис. 1.4. Методология Инмона — нисходящий дизайн, в котором сначала строится централизованное хранилище данных, а затем на его основе создаются витрины данных

Многие практикующие специалисты с осторожностью относятся к использованию нормализованных структур на уровне интеграции и пространственных структур для целей представления. Это связано с тем, что данные извлекаются, преобразуются и загружаются дважды. Сначала данные извлекаются, преобразуются и загружаются на нормализованном уровне интеграции. Затем все это делается заново для итоговой загрузки данных в многомерную модель. Разумеется, этот двушаговый процесс требует больше времени и ресурсов для разработки, больше времени для периодической загрузки или обновления данных и больше пространства для хранения копий данных. Другой недостаток заключается в том, что при добавлении новых данных в витрину их придется сначала добавить на уровень интеграции. Так как разработка требует времени, а изменения на уровне интеграции — крайней осторожности, пользователям, запрашивающим новые данные, придется дольше ждать.

Более того, избыточность — дублирование данных, без которого можно обойтись, — часто преподносится как проблема на уровне интеграции Инмона. Тем не менее в эпоху облачных вычислений этот аргумент уже не столь актуален. Облачные хранилища стали довольно дешевы, тогда как вычисления могут оставаться затратными. Из-за этих высоких затрат многие эксперты сейчас предпочитают метод моделирования данных Кимбалла.

---

[1] Здесь речь идет о «сборке обратно» бизнес-объектов согласно бизнес-доменам для того, чтобы на них потом удобнее было смотреть аналитикам. — *Примеч. науч. ред.*

Методология Кимбалла

Методология Кимбалла, названная по имени ее создателя Ральфа Кимбалла (Ralph Kimball), была представлена в 1996 году как метод моделирования данных и часто используется при организации хранилищ данных[1]. Она ориентируется на создание таблиц измерений для эффективной аналитической обработки. При таком подходе сначала создаются размерные витрины данных для реакции на потребности бизнеса. Для этого Кимбалл рекомендует применять метод размерного моделирования с использованием схемы «звезда».

Абстрактное представление методологии Кимбалла изображено на рис. 1.5.

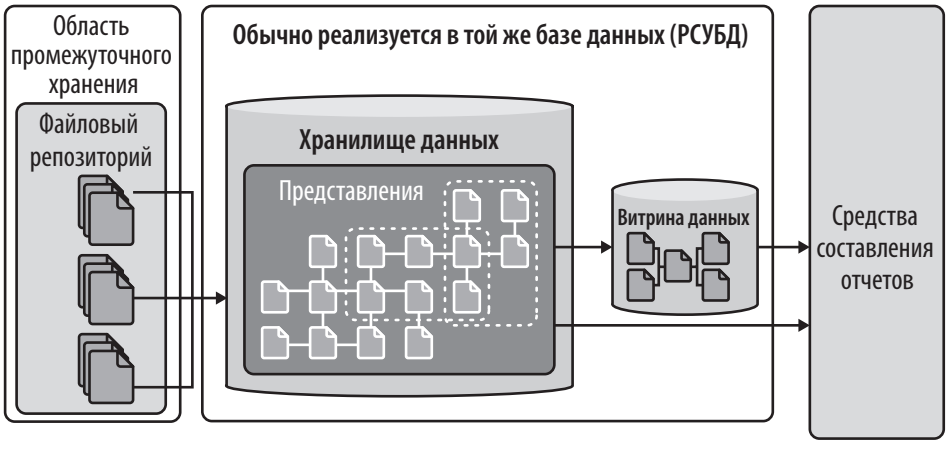

Рис. 1.5. Методология Кимбалла — восходящий метод построения хранилищ данных

В этом подходе к моделированию данных уровень интеграции хранилища данных рассматривается как конгломерат или набор таблиц измерений (dimension tables), которые являются производными копиями транзакционных данных из систем-источников. Переданные на уровень интеграции данные уже оптимизированы для быстрого чтения. Эти данные, более плоские и разреженные, сильно напоминают структуры витрин данных в методологии Инмона. Однако в отличие от модели Инмона уровень интеграции у Кимбалла включает таблицы измерений, образующие основу для витрин данных. Таким образом, модель Кимбалла не только признает существование витрин данных, но и рассматривает их как необходимые для повышения производительности и создания подвыборок. Витрины данных допускают агрегирование или изменение хранимых копий данных на основании требований групп пользователей. Интересно, что витрины данных также могут

[1] Ральф Кимбалл представил концепцию размерного моделирования для отрасли хранилищ данных / бизнес-аналитики в 1996 году в своей авторитетной книге The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses (John Wiley & Sons).

быть виртуальными. Это логические представления с размерными моделями, построенные на основе существующих таблиц измерений и таблиц фактов (fact tables) на уровне интеграции, обеспечивающие гибкость и эффективность обработки данных.

Путаница с функциями уровней хранилищ данных

Важно пояснить, что с функциями уровней хранилищ данных иногда возникает путаница, такая же, как та, что может возникнуть при формировании уровней архитектуры медальона. Эти разные уровни, образующие среднюю часть общей большой архитектуры, проектируются для закрепления заметно различающихся обязанностей за разными этапами, соответствующими распространенным практикам в архитектуре ПО. Как правило, присутствует уровень промежуточного хранения (staging) или загрузки (ingestion), на котором хранятся сырые данные; этот уровень фактически отделяет системы-источники от хранилища данных. За ним следует уровень интеграции или трансформации, на котором данные интегрируются после выполнения всех приемочных критериев области промежуточного хранения. Здесь очищенные, исправленные, обогащенные и преобразованные данные хранятся в унифицированной модели. Эта модель гармонизирована, что означает, что форматы, типы, имена, структуры и отношения в ней стандартизированы. Уровень также содержит исторические данные, обрабатываемые для получения информации об изменениях во времени. Наконец, присутствует уровень представления (или презентационный уровень), на котором выбираются данные, актуальные для конкретных сценариев использования. Модель данных подстраивается под специфические требования сценариев использования.

Однако важно отметить возможные веские причины отхода от этой традиционной трехуровневой архитектуры хранилища данных. Ради гибкости или возможности аудита некоторые организации реализуют дополнительные уровни. Например, можно добавить дополнительный уровень для аудируемости, где источники сначала связываются с целевой моделью перед слиянием с другими источниками. Также область промежуточного хранения может быть разбита на дешевый файловый репозиторий со всеми поставленными данными и реляционную базу данных, содержащую только самые последние проверенные данные. Ключевой вывод: количество уровней или зон зависит от ваших требований. Единственно правильного ответа не существует. Все сводится к выбору подходящих для ваших целей компромиссов.

Чтобы облегчить размерное моделирование таким образом, Кимбалл ввел концепцию *согласованных измерений* (conformed dimensions). Это ключевые измерения,

которые становятся общими и используются разными группами пользователей. Кроме того, Кимбалл вводит метод историзации данных с использованием SCD (Slowly Changing Dimensions, медленно меняющиеся измерения).

SCD представляют собой таблицы, в которых медленно и планомерно сохраняются все исторические изменения. Другими словами, SCD — разновидность измерения с атрибутами, отражающими изменение во времени. SCD1, SCD2 и SCD3 — самые распространенные методы для работы с такими изменениями:

SCD1 (https://oreil.ly/yjJ8c)

В этой разновидности, также называемой перезаписью (overwrite), существующие записи в хранилище данных просто обновляются свежей информацией. Этот метод хорошо подходит для сценариев, когда исторические данные не важны и значение имеют только самые актуальные данные. При этом SCD типа 1 не позволяет отслеживать исторические изменения, так как старые данные перезаписываются новыми.

SCD2 (https://oreil.ly/XfL43)

В этой разновидности, также называемой «добавлением новой строки», в хранилище данных создается новая запись для каждого происходящего изменения с сохранением исходной записи. Этот метод полезен, если исторические данные важны и их необходимо хранить. Для отслеживания изменений со временем в SCD2 создается новая запись с новым значением первичного ключа, но исходная запись при этом сохраняется с отдельным значением первичного ключа. Так хранилище данных может поддерживать полную историю изменений во времени.

SCD3 (https://oreil.ly/3IFm9)

В этой разновидности, также называемой добавлением нового атрибута, в существующую запись в хранилище данных добавляется новый атрибут для отслеживания изменений. Этот метод хорошо подходит, если требуется отслеживать изменение во времени только для небольшого количества атрибутов. Однако у SCD3 есть ограничения — прежде всего то, что этот метод позволяет отслеживать лишь ограниченный объем исторических данных (обычно только одно предыдущее состояние).

В эпоху современных архитектур данных моделирование данных (включая методологии Инмона и Кимбалла) продолжает играть важнейшую роль в управлении данными и эффективном использовании их потенциала. Оно помогает понять и грамотно применять сложные данные за счет разделения обязанностей загрузки, интеграции/гармонизации и потребления. Создание надежных представлений данных и их взаимоотношений упрощает использование данных техническими и нетехническими стейкхолдерами. Кроме того, моделирование данных способствует повышению производительности и оптимизации запросов. С хорошо структурированной моделью проще найти и получить конкретные данные, а это

повышает скорость работы и производительность системы. Наконец, такая модель поддерживает более высокие уровни управления данными и безопасности. С четко выстроенными моделями организации могут реализовать улучшенные политики управления данными, тем самым обеспечивая требуемый контроль доступа к данным и их правильное использование.

Традиционные хранилища данных: ключевые выводы

На этом наше обсуждение традиционных хранилищ данных подходит к концу. Мы рассмотрели методологии Инмона и Кимбалла, которые остаются актуальными до сих пор. Ниже приведены основные выводы касательно традиционных хранилищ данных.

Во-первых, концепция разделения данных на уровни не нова. Она доказала свою эффективность как стратегию разделения обязанностей, способствующую более эффективной организации данных и управлению ими.

Во-вторых, моделирование данных чрезвычайно важно. Оно играет значительную роль в обеспечении гибкости, сокращении избыточности данных и повышении производительности, а также служит интерфейсом для бизнеса. Правильная организация моделирования данных — важнейшая составляющая эффективности любой системы управления данными.

Наконец, традиционные хранилища данных обеспечивают тесную интеграцию между программной и аппаратной частями. В таких системах, обычно размещаемых локально, вычислительные ресурсы интегрируются с пространством хранения, в результате чего работа с данными проходит быстро и эффективно. В эти системы входят сложные программы, повышающие до максимума производительность оборудования, на котором они выполняются. Эти системы могут вертикально масштабироваться за счет дополнительной физической инфраструктуры. Несмотря на потенциальную дороговизну и ограничения, они когда-то были предпочтительным вариантом для многих организаций и продолжают приносить пользу в специализированных сценариях.

Хранилища данных обладают огромной ценностью для бизнеса, поскольку предоставляют качественные стандартизированные данные, необходимые для принятия обоснованных решений. Залогом их эффективности становится экспертное моделирование данных и тесная интеграция оборудования и хранилища, обеспечивающая быструю выборку данных. Все это делает их важнейшим инструментом бизнес-операций.

Тем не менее традиционные архитектуры хранилищ данных, использующие реляционные системы управления БД (РСУБД), сталкиваются с затруднениями при обработке быстро растущих объемов данных. У них появляются проблемы с объемом хранилища и масштабируемостью, которые могут приводить к значительным затратам. Главная проблема заключается в том, что вертикальное масштабирование (наращивание мощности отдельной машины) имеет свои пределы и может

обходиться дорого. Помимо затрат есть и другие проблемы, препятствующие масштабированию архитектуры хранилищ данных для удовлетворения текущих потребностей, в частности недостаточная гибкость при поддержке разных типов рабочих нагрузок, таких как работа с неструктурированными данными и решение задач машинного обучения. По этой причине инженеры начали исследовать другие архитектуры, которые позволяют решать эти проблемы. Это подводит нас к следующему разделу, в котором будут рассматриваться архитектуры озер данных.

Краткая история озер данных

Концепция озер данных (data lakes) появилась как решение, исправляющее недостатки традиционных хранилищ данных. Озера данных стали набирать популярность в середине 2000-х, одновременно с расцветом программных проектов с открытым исходным кодом. В отличие от своих предшественников, озера данных ввели новую распределенную архитектуру, способную управлять огромными объемами данных в разных состояниях: неструктурированном, полуструктурированном и структурированном. Такая гибкость расширила возможности практического использования данных.

Озера данных используют ПО с открытым исходным кодом и, как следствие, могут работать на любом стандартном или бюджетном оборудовании потребительского класса. Этот отход от проприетарных РСУБД ознаменовал значительные изменения в отношении построения решений Big Data и отказ от дорогостоящих аппаратных кластеров. Кроме того, интеграция технологий машинного обучения в озера данных наделяет их возможностями, выходящими за рамки традиционного составления отчетов в хранилищах данных. Чтобы получить наглядное представление о структуре озер данных, взгляните на диаграмму на рис. 1.6.

Первое поколение озер данных в основном строилось на базе Hadoop — известного фреймворка с открытым исходным кодом, включающего различные инструменты и сервисы. В основе Hadoop лежит программный фреймворк MapReduce и файловая система HDFS (Hadoop Distributed File System)[1], предоставляющая возможность обработки больших датасетов с использованием распределенных алгоритмов в кластере. Кроме того, Hadoop включает ряд других средств, расширяющих его возможности хранения, обработки и анализа огромных объемов данных.

---

[1] Первоисточником Hadoop стала статья Санджая Гемавата (Sanjay Ghemawat), Говарда Гобьофа (Howard Gobioff) и Шун-Так Люна (Shun-Tak Leung) The Google File System (https://oreil.ly/rLCCa), опубликованная в 2003 году. За ней последовала вторая влиятельная статья MapReduce: Simplified Data Processing on Large Clusters (https://oreil.ly/lqVkL) Джеффри Дина (Jeffrey Dean) и Санджая Гемавата.