

ГЛАВА 1

Компромиссы в архитектуре систем обработки данных

Не существует решений, есть только компромиссы. <...> Но вы стараетесь добиться наилучшего результата, и это все, на что вы можете надеяться.

*Томас Соуэлл,
из интервью Фреду Барнсу (2005)*

В настоящее время данные играют главенствующую роль в разработке большинства приложений. В мобильных и веб-приложениях, программном обеспечении как услуге (software as a service, SaaS) и облачных сервисах стало нормой хранить данные множества разных пользователей в общей серверной инфраструктуре. Данные об активности пользователей, транзакциях, устройствах и датчиках необходимо хранить и предоставлять для анализа. Взаимодействуя с приложением, пользователи не только читают хранящиеся данные, но и генерируют новые.

Небольшие объемы данных, которые можно хранить на одном компьютере, довольно легко обрабатывать. Однако с ростом этого объема или частоты запросов возникает необходимость распределения обработки между несколькими компьютерами, что создает множество проблем. С усложнением потребностей приложения уже недостаточно хранить все данные в одной системе и может понадобиться объединить несколько систем хранения, обладающих различными возможностями.

Мы называем приложение *высоконагруженным данными* (data-intensive applications, DIA), если управление данными — одна из основных сложностей,

которые должны преодолеть разработчики [1]. В отличие от приложений, *высоконагруженных вычислениями* (compute-intensive), где главная сложность заключается в распараллеливании больших объемов вычислений, в *высоконагруженных данными* приложениях гораздо больше беспокойства доставляют такие аспекты, как хранение и обработка больших объемов данных, управление изменениями данных, обеспечение согласованности при сбоях и конкурентном доступе, а также высокой доступности сервисов.

Подобные приложения обычно создаются из стандартных блоков, обеспечивающих часто требующуюся функциональность. Например, многим приложениям нужно:

- хранить данные, чтобы эти или другие приложения могли найти их в дальнейшем (*базы данных*);
- запоминать результат ресурсоемкой операции для ускорения чтения (*кэши*);
- предоставлять пользователям возможность искать данные по ключевому слову или фильтровать их различными способами (*поисковые индексы*);
- обрабатывать события и данные немедленно, по мере их поступления (*поточковая обработка*);
- время от времени перемалывать большие объемы накопленных данных (*пакетная обработка*).

При разработке приложения мы обычно берем несколько программных систем или сервисов, таких как базы данных или API, и объединяем их с помощью своего прикладного кода. Если вы делаете именно то, для чего были разработаны эти системы обработки данных, то процесс может оказаться довольно простым.

Однако по мере развития приложения неизбежно возникают проблемы. Существует множество систем баз данных с разнообразными характеристиками, подходящих для разных целей, — как выбрать наиболее подходящую для приложения? Есть много подходов к кэшированию, несколько способов построения поисковых индексов и т. д. — как оценить их достоинства и недостатки? Необходимо определить, с помощью каких инструментов и подходов лучше всего решать поставленную задачу. Кроме того, иногда бывает непросто подобрать нужную комбинацию инструментов, когда нужно сделать что-то, для чего одного инструмента в отдельности недостаточно.

Эта книга учит принимать решения при выборе технологий и демонстрирует приемы их комбинирования. Как вы увидите сами, ни один подход не является принципиально лучшим, чем другие, — у всех есть свои плюсы и минусы. Эта книга научит вас задавать правильные вопросы для оценки и сравнения систем обработки данных, чтобы определить, какой подход лучше всего удовлетворит потребности конкретного приложения.

Свое путешествие мы начнем со знакомства с некоторыми типичными способами использования данных в современных организациях. Многие из представленных здесь идей берут свое начало в *корпоративных системах* (то есть в программном обеспечении и инженерных практиках крупных организаций, корпораций и государственных учреждений), поскольку исторически только крупные организации имели большие объемы данных, обработка которых требует сложных технических решений. Если объем данных довольно мал, то их можно хранить даже в простой электронной таблице! Однако в последнее время и в небольших компаниях стала появляться потребность управлять большими объемами данных и создавать системы, высоконагруженные данными.

Одна из основных проблем, характерных для систем обработки данных, заключается в том, что разным людям приходится решать совершенно разные задачи. У каждой команды внутри компании могут быть различные цели, даже если они работают с одним и тем же набором данных! Более того, эти цели могут быть сформулированы довольно расплывчато, что способно привести к недопониманию и разногласиям при выборе подхода к решению.

Чтобы помочь вам разобраться в имеющихся вариантах, в этой главе сравниваются несколько концепций и рассматриваются их компромиссы. Далее в этой главе мы уделим внимание таким темам, как:

- разница между оперативными и аналитическими системами (раздел «Оперативные и аналитические системы»);
- достоинства и недостатки облачных сервисов и локально размещаемых систем (раздел «Размещение в облаке и локально»);
- признаки необходимости перехода от систем с одним узлом к распределенным системам (раздел «Распределенные и одноузловые системы»);
- учет потребностей бизнеса и прав пользователя (раздел «Системы обработки данных, право и общество»).

В этой главе также даются определения терминов, широко используемых во всей книге.

ТЕРМИНОЛОГИЯ: ФРОНТЕНДЫ И БЭКЕНДЫ

Большая часть книги посвящена разработке *бэкенда*. В веб-приложениях клиентский код, который выполняется в браузере, называется *фронтендом* (*frontend*), а серверный, обрабатывающий запросы пользователей, — *бэкендом* (*backend*). Мобильные приложения похожи на фронтенды — они предоставляют пользовательский интерфейс, нередко взаимодействующий через Интернет с серверным приложением — бэкендом. Иногда фронтенды управляют локальными данными, находящимися на устройстве пользователя [2], но наибольшие проблемы с инфраструктурой данных обычно возникают в бэкенде: фронтенд должен обрабатывать данные только одного пользователя, тогда как бэкенд управляет данными *всех* пользователей.

Доступ к бэкенду (серверной части) часто осуществляется по протоколу HTTP (иногда WebSocket). Обычно бэкенд состоит из прикладного кода, который читает данные из одной или нескольких баз данных и записывает в них и иногда взаимодействует с дополнительными системами данных, такими как кэши или очереди сообщений (все вместе их можно назвать *инфраструктурой данных*). Прикладной код часто не хранит текущего состояния (то есть по завершении обработки одного HTTP-запроса он забывает все, что было с ним связано), и любая информация, которая должна сохраняться между запросами, должна оставаться либо на стороне клиента, либо в серверной инфраструктуре данных.

Оперативные и аналитические системы

Работающие с системами обработки данных на предприятиях нередко сталкиваются с несколькими группами специалистов, использующих данные. Первая — *бэкенд-инженеры*, которые создают сервисы, обрабатывающие запросы на чтение и обновление данных. Эти сервисы часто обслуживают внешних пользователей напрямую или косвенно через другие сервисы (см. подраздел «Микросервисы и бессерверные вычисления» далее). Некоторые сервисы могут предназначаться исключительно для внутреннего применения другими подразделениями организации.

Помимо команд, управляющих сервисами, доступ к данным организации обычно требуется еще двум группам работников: *бизнес-аналитикам*, создающим отчеты о деятельности организации, чтобы помочь руководству принимать более обоснованные решения, и *специалистам по анализу данных*, которые ищут в данных новые идеи или новые способы их представления пользователям,

реализуемые с помощью анализа данных и машинного обучения (МО)/искусственного интеллекта (ИИ). Примером могут служить генерируемые рекомендации («люди, купившие X , также покупают Y ») на сайте интернет-магазина, прогнозная аналитика (оценка рисков), фильтрация спама и ранжирование результатов поиска.

Как правило, бизнес-аналитики и специалисты по анализу данных используют разные инструменты и работают по-разному, но у них есть некоторые общие практики. Во-первых, и те и другие занимаются *анализом*, то есть изучают данные, сгенерированные пользователями и сервисами. Во-вторых, они обычно не изменяют данные (за исключением, может быть, исправления ошибок), но могут создавать производные наборы данных, включающие исходные данные, обработанные некоторым образом.

В результате системы разделились на два типа, и это различие мы будем использовать в книге.

- *Оперативные системы* (operational systems) состоят из сервисов и инфраструктуры данных и создают данные, например, в процессе обслуживания внешних пользователей. Прикладной код в этих системах не только читает, но и изменяет данные в своих хранилищах в зависимости от действий пользователей.
- *Аналитические системы* (analytical systems) удовлетворяют потребности бизнес-аналитиков и специалистов по обработке данных. Они содержат копию данных из оперативных систем, используемую только для чтения, и оптимизированы для вариантов обработки данных, необходимых аналитикам.

Как будет показано в следующем разделе, оперативные и аналитические системы часто разделяются, и на то есть веские причины. По мере развития этих систем появились две новые специализированные роли: инженеры данных и инженеры-аналитики. *Инженеры данных* (data engineers) умеют интегрировать оперативные и аналитические системы и несут ответственность за инфраструктуру данных организации в целом [3]. *Инженеры-аналитики* (analytics engineers) моделируют и преобразуют данные, чтобы сделать их более полезными для бизнес-аналитиков и специалистов по обработке данных в организации [4].

Многие инженеры специализируются либо на оперативной, либо на аналитической стороне вопроса. Однако эта книга охватывает и оперативные, и аналитические системы обработки данных, поскольку и те и другие занимают важное положение в жизненном цикле данных в организации. Мы подробно рассмотрим инфраструктуру данных для предоставления услуг как внутренним, так и внешним пользователям, чтобы вы могли совершенствовать взаимодействие со своими коллегами, находящимися по другую сторону этой границы.

Характеристики транзакционной и аналитической обработки

На заре обработки бизнес-данных запись в базу данных обычно соответствовала совершению бизнес-операции — транзакции: продаже, размещению заказа у поставщика, выплате заработной платы сотруднику и т. д. Однако даже после проникновения баз данных в области, не связанные с денежными операциями, термин «транзакция» закрепился и в настоящее время обозначает группу операций чтения и записи, объединенных в логическую единицу.



В главе 8 мы подробно рассмотрим, что подразумеваем под транзакцией. В текущей главе этот термин используется в более широком смысле и обозначает операции чтения и записи с низкой задержкой.

Несмотря на то что базы данных стали применяться для обработки самых разных данных — сообщений в социальных сетях, результатов игр, контактов в адресной книге и многого другого, основная схема доступа оставалась схожей с обработкой бизнес-транзакций. Оперативная система обычно выполняет поиск небольшого количества записей по ключу (это называется *точечным запросом*). Записи вставляются, обновляются или удаляются в зависимости от того, что вводит пользователь. Поскольку эти приложения интерактивные, такая схема доступа получила название *оперативной обработки транзакций* (online transaction processing, OLTP).

Однако постепенно базы данных стали все чаще использоваться для аналитики, характеризующейся совершенно иными моделями доступа, отличными от OLTP. Обычно аналитический запрос не возвращает отдельные записи, а сканирует огромное количество записей и вычисляет сводные статистические данные, такие как количество, сумма или среднее значение. Например, бизнес-аналитик в сети супермаркетов может захотеть ответить на такие аналитические запросы.

- Какой была общая выручка каждого из магазинов в январе?
- На сколько больше обычного было продано бананов во время последней рекламной акции?
- Какую марку детского питания чаще всего покупают вместе с подгузниками марки X?

Отчеты, получаемые в ответ на такие запросы, играют важную роль, так как помогают руководству принимать решения о дальнейших действиях. Чтобы отличить этот способ использования баз данных от обработки транзакций, его называют *оперативной аналитической обработкой* (online analytical processing, OLAP) [5]. Разница между OLTP и аналитикой не всегда очевидна, однако есть некоторые типичные характеристики, которые перечислены в табл. 1.1.

Таблица 1.1. Характеристики оперативных и аналитических систем

Характеристика	Оперативные системы	Аналитические системы
Основная модель чтения	Точечные запросы (получение отдельных записей по ключу)	Агрегирование по большому количеству записей
Основная модель записи	Создание, обновление и удаление отдельных записей	Массовый импорт (ETL) или поток событий
Пример пользователя-человека	Конечный пользователь мобильного или веб-приложения	Внутренний аналитик, обеспечивающий поддержку при принятии решений
Пример пользователя-машины	Проверка авторизации	Выявление мошеннических схем/злоупотреблений
Типы запросов	Фиксированные, определяемые потребностями приложения	Произвольные, несистематические исследования, проводимые аналитиками
Количество запросов	Много простых запросов	Мало запросов, но все они сложные
Что представляют данные	Текущее состояние (на данный момент времени)	Историю событий, произошедших за определенный период времени
Размер набора данных	От гигабайтов до терабайтов	От терабайтов до петабайтов



Значение термина «оперативный» (online в аббревиатуре OLAP) не совсем ясно. Вероятно, он указывает не только на то, что запросы предназначены для заранее определенных отчетов, но и на возможность использования системы OLAP в интерактивном режиме для выполнения исследовательских запросов.

В оперативных системах пользователям, как правило, не разрешается создавать собственные SQL-запросы и выполнять их в базе данных, потому что это позволило бы им читать или изменять данные, к которым у них нет доступа. Они также могут писать ресурсоемкие запросы и, следовательно, негативно влиять на общую производительность базы данных. По этим причинам в OLTP-системах в основном выполняются фиксированные наборы запросов, встроенные в прикладной код, а разовые пользовательские запросы задействуются лишь изредка — для обслуживания или устранения неполадок. В то же время аналитические базы данных обычно позволяют пользователям писать произвольные SQL-запросы вручную или генерировать запросы автоматически с помощью инструментов визуализации и панелей мониторинга, таких как Tableau, Looker или Microsoft Power BI.

Системы еще одного типа, предназначенные для решения аналитических задач (выполнения запросов, агрегирующих множество записей), но встроенные в продукты, ориентированные на пользователя, известны как системы *продуктовой аналитики* (product analytics) или *аналитики реального времени*. К ним

относятся, например, Pinot, Druid и ClickHouse [6]. Такие системы обрабатывают данные в реальном времени и оптимизированы для получения ответов на запросы с низкой задержкой. В отличие от них традиционные OLAP-системы обычно обрабатывают данные пакетами и оптимизированы для обработки запросов с высокой пропускной способностью.

Склады данных

Первоначально одни и те же базы данных использовались как для обработки транзакций, так и для выполнения аналитических запросов. Язык SQL оказался достаточно гибким в этом отношении. Он хорошо подходит для запросов обоих типов. Однако в конце 1980-х и начале 1990-х возникла тенденция к тому, что компании перестали использовать свои OLTP-системы для аналитических целей и начали применять для этого отдельные системы баз данных. Эти отдельные базы данных получили название «склады данных»¹ (data warehouse).

Крупное предприятие может иметь десятки и даже сотни OLTP-систем, обеспечивающих работу веб-сайта, ориентированного на клиентов. Они управляют системами учета продаж (кассовыми аппаратами) в физических магазинах, отслеживают запасы на складах, планируют маршруты для перевозки товаров, управляют поставщиками, администрируют персонал и выполняют множество других задач. Каждая из этих систем сложна, и для их обслуживания требуются отдельные команды специалистов, поэтому они работают практически независимо друг от друга.

Как правило, бизнес-аналитикам и специалистам по обработке данных нежелательно напрямую обращаться к этим OLTP-системам по нескольким причинам.

- Данные, представляющие интерес, могут быть разбросаны по нескольким оперативным системам, что затрудняет их сведение в одном запросе (проблема, известная как *разрозненные хранилища данных*).
- Схемы и структуры данных, подходящие для OLTP, менее пригодны для аналитики (см. раздел «Звезды и снежинки: схемы для аналитики» главы 3).
- Аналитические запросы могут быть довольно ресурсоемкими и отрицательно сказываться на общей производительности баз данных OLTP.
- Системы OLTP могут располагаться в отдельной сети, к которой запрещен прямой доступ по соображениям безопасности или соответствия нормативным требованиям.

¹ В русскоязычной литературе термин data warehouse часто переводится как «хранилище данных». Однако он имеет слишком широкое толкование, поэтому здесь и далее для перевода data warehouse будет использоваться термин «склад данных», а под хранилищем данных будет подразумеваться обобщенная система хранения данных (storage data). — *Примеч. пер.*

Склад данных — это отдельная база данных, к которой аналитики могут обращаться по своему усмотрению, не влияя на деятельность OLTP [7]. Как будет показано в главе 4, склады данных часто хранят данные совершенно иначе, чем базы данных OLTP, чтобы оптимизировать их для разных видов запросов, распространенных в аналитике.

Склад данных содержит копию данных из различных OLTP-систем компании, доступную только для чтения. Данные извлекаются из баз данных OLTP (либо путем периодической выгрузки, либо передачей непрерывного потока обновлений), преобразуются в схему, удобную для анализа, очищаются, а затем загружаются в склад данных. Этот процесс известен как «*извлечение — преобразование — загрузка*» (extract — transform — load, ETL) (рис. 1.1). Иногда этапы *преобразования* и *загрузки* меняются местами, то есть преобразование выполняется в складе данных после загрузки, что дает процесс «*извлечение — загрузка — преобразование*» (extract — load — transform, ELT).

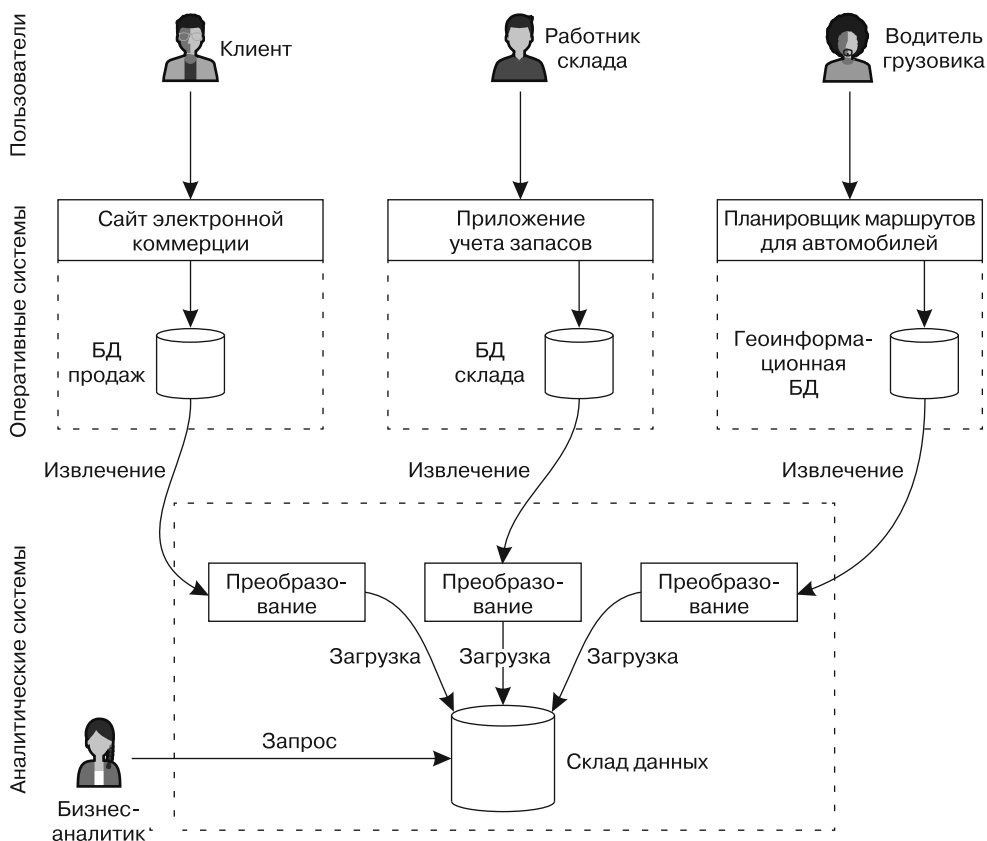


Рис. 1.1. Упрощенная схема извлечения, преобразования и загрузки информации в склад данных

В некоторых случаях источниками данных для процессов ETL являются внешние SaaS-продукты, например системы автоматизации взаимодействий с клиентами (customer relationship management, CRM), электронного маркетинга или обработки платежей по кредитным картам. В таких ситуациях доступ к исходной базе данных возможен только через API поставщика программного обеспечения. Перенос данных из этих внешних систем позволяет выполнять виды анализа, которые невозможно реализовать через SaaS API. Процесс ETL для SaaS API часто реализуется с помощью специализированных сервисов передачи данных, таких как Fivetran, Singer или Airbyte.

Некоторые системы баз данных предлагают *гибридную транзакционно-аналитическую обработку* (hybrid transactional/analytical processing, HTAP), которая направлена на обеспечение OLTP и аналитики в одной системе без необходимости выполнения процедур ETL для передачи информации из одной системы в другую [8], [9]. Однако многие системы HTAP содержат систему OLTP, связанную с отдельной аналитической системой, скрытой за общим интерфейсом, поэтому важно понимать, как работают эти системы.

Более того, несмотря на существование HTAP, транзакционные и аналитические системы продолжают разделять — из-за разных целей и требований. В частности, считается хорошей практикой, когда каждая оперативная система имеет собственную базу данных (см. подраздел «Микросервисы и бессерверные вычисления» далее), что может привести к появлению сотен отдельных оперативных баз данных. В то же время предприятие обычно имеет единый склад данных, чтобы бизнес-аналитики могли объединять данные из нескольких оперативных систем в одном запросе.

Иными словами, HTAP не заменяет склад данных. Скорее, она полезна, когда одному и тому же приложению необходимо параллельно с аналитическими запросами, сканирующими большое количество записей, читать и обновлять отдельные записи с низкой задержкой. Такой характер нагрузки может иметь, например, система обнаружения мошенничества [10].

Разделение оперативных и аналитических систем — это часть более общей тенденции. По мере того как рабочие нагрузки усложняются, системы становятся все более специализированными и оптимизированными для конкретных задач. Системы общего назначения могут без проблем обрабатывать небольшие объемы данных, но чем больше масштаб, тем более специализированными становятся системы [11].

От складов к озерам данных

В складах данных часто используется *реляционная* модель, запросы к которой выполняются с помощью SQL (см. главу 3) и нередко с применением специализированного ПО для бизнес-аналитики. Эта модель хорошо подходит для

запросов, востребованных бизнес-аналитиками, но она не так удобна специалистам по анализу, выполняющим задачи, перечисленные далее.

- Преобразование данных в форму, подходящую для обучения моделей МО. Часто это требует преобразования записей и столбцов из таблиц базы данных в вектор или матрицу числовых значений, называемых *признаками*. Такое преобразование, предназначенное для достижения максимальной производительности обученной модели, называется *проектированием признаков* и обычно требует писать уникальный код, который сложно выразить с помощью SQL.
- Использование методов обработки естественного языка (natural language processing, NLP) для анализа текстовых данных, допустим отзывов о товаре, для извлечения из них структурированной информации, например эмоционального состояния автора или упомянутых им тем. А еще специалистам по анализу данных может потребоваться извлечь структурированную информацию из фотографий с помощью методов компьютерного зрения.

Несмотря на попытки добавить операторы МО в модель данных SQL [12] и создать эффективные системы МО на основе реляционных баз данных [13], многие специалисты по анализу данных предпочитают не работать с реляционными базами данных, а использовать библиотеки на Python, такие как Pandas и scikit-learn, языки статистического анализа, например R, и распределенные аналитические фреймворки, такие как Spark [14]. Мы подробнее обсудим этот аспект в разделе «Матрицы, массивы и таблицы DataFrame» главы 3.

Как следствие, организации сталкиваются с необходимостью предоставлять данные в форме, подходящей для использования специалистами по анализу. Решением является *озеро данных* (data lake) — централизованное хранилище, где находятся копии любых данных, которые были получены с применением процессов ETL из оперативных систем и могут быть полезны для анализа. В отличие от склада данных озеро просто хранит файлы, не навязывая какой-то конкретный формат, модель данных или схему [15]. Файлы, хранящиеся в озере данных, могут содержать наборы записей из базы данных, закодированные с использованием формата файлов, такого как Avro или Parquet (см. главу 5), также озеро данных может хранить такие данные, как текст, изображения, видео, показания датчиков, разреженные матрицы, векторы признаков, последовательности генома и вообще любого типа [16]. Помимо большей гибкости, озеро данных часто оказывается дешевле в эксплуатации, чем реляционное хранилище данных, потому что может задействовать стандартное хранилище файлов, такое как объектные хранилища (см. подраздел «Архитектура облачно-нативных систем» далее).

Процессы ETL были обобщены до *конвейеров данных*, и в некоторых случаях озеро данных стало промежуточной остановкой на пути от оперативных систем

к складу данных. Озеро хранит данные в исходном виде, в котором они были получены из оперативных систем, не преобразуя их в реляционную схему. Преимущество такого подхода заключается в том, что каждый потребитель может преобразовать исходные данные в форму, которая наилучшим образом соответствует его потребностям. Иногда это называют *принципом суши*: «сырые данные лучше» [17].

За пределами озера данных

По мере развития аналитики организации все больше внимания уделяют управлению аналитическими системами и конвейерами данных и их эксплуатации, что отражено, например, в *DataOps Manifesto* [18]. Отчасти это обусловлено необходимостью совершенствования управления, укрепления конфиденциальности и соблюдения таких нормативных актов, как Общий регламент по защите данных (General Data Protection Regulation, GDPR) и Закон Калифорнии о защите конфиденциальности потребителей (California Consumer Privacy Act, CCPA), которые обсуждаются в разделе «Системы обработки данных, право и общество» далее в этой главе и в главе 14.

Еще один важный фактор: данные для аналитики все чаще предоставляются не только в виде файлов и реляционных таблиц, но и в виде потоков событий (см. главу 12). При хранении данных в файлах анализ для учета изменений можно запускать через определенные периоды времени, например раз в сутки, но потоковая обработка позволяет аналитическим системам реагировать на события гораздо быстрее, буквально в течение нескольких секунд. В зависимости от приложения и его чувствительности к времени подход с использованием потоковой обработки может пригодиться, например, для выявления и блокировки потенциально мошеннической или противоправной деятельности.

В некоторых случаях результаты работы аналитических систем становятся доступными оперативным системам (иногда этот процесс называют *обратным ETL* [19]). Так, модель МО, обученная на данных из аналитической системы, может быть развернута в производственной среде и генерировать рекомендации для конечных пользователей, например: «Люди, купившие X, также покупают Y». Модели машинного обучения могут разворачиваться в оперативных системах с помощью специализированных инструментов, таких как TFX, Kubeflow или MLflow.

Системы учета и производные данные

Из-за различий между оперативными и аналитическими системами в этой книге также проводится различие между *системами учета* (systems of record) и *системами производных данных* (derived data systems). Эти термины полезны тем, что помогают прояснить поток данных через систему.

Системы учета

Также известны как *источники истины*, хранят авторитетную, или *каноническую*, версию данных. Когда поступают новые данные, например введенные пользователем, они сначала записываются в систему учета. Каждый факт представлен ровно один раз (представление обычно *нормализовано*; см. подраздел «Нормализация, денормализация и соединения» в главе 3). Если между другой системой и системой учета имеется какое-либо расхождение, то верным считается значение в системе учета (по определению).

Системы производных данных

В них данные являются результатом преобразования или обработки данных, полученных из другой системы. В случае потери производные данные можно воссоздать из исходного источника. Классический пример — кэш: данные могут быть предоставлены из кэша, если они там имеются, в противном случае можно обратиться к исходной базе данных. К этой категории относятся также денормализованные значения, индексы, материализованные представления, преобразованные представления данных и модели, обученные на наборе данных.

С технической точки зрения производные данные *избыточны* — они фактически дублируют существующую информацию. Однако зачастую они необходимы для увеличения производительности запросов на чтение. Из одного источника можно создать несколько наборов производных данных, что позволяет рассматривать данные с разных точек зрения.

Аналитические системы обычно являются системами производных данных, потому что потребляют данные, созданные в других местах. Оперативные сервисы могут включать комбинации систем учета и систем производных данных. Системы учета — это первичные базы данных, куда записываются исходные данные, а системы производных данных — это индексы и кэши, которые ускоряют операции чтения, особенно для запросов, на которые система учета не может эффективно ответить.

Большинство баз данных, механизмов хранения и языков запросов сами по себе не являются системами учета или системами производных данных. Они лишь инструмент, и от вас зависит, как он будет использоваться. Различие между системой учета и системой производных данных определяется не инструментом, а способом его применения. Четко определив, какие данные являются исходными, а какие — производными, можно внести ясность в запутанную архитектуру системы.

Когда данные, хранящиеся в одной системе, получают из данных, хранящихся в другой, необходим процесс обновления данных при изменении

исходных данных в первичной системе учета. К сожалению, многие базы данных разработаны на основе предположения, что приложение всегда будет использовать только одну базу данных, и они никак не упрощают интеграцию нескольких систем для распространения таких обновлений. В главе 11 мы обсудим конвейеры данных как подход к *интеграции данных*, который позволяет объединить несколько систем данных для достижения целей, недоступных для одной системы.

На этом мы завершаем сравнение аналитики и обработки транзакций и в следующем разделе рассмотрим еще один компромисс, который, возможно, уже неоднократно обсуждался.

Размещение в облаке и локально

При решении любой задачи один из первых вопросов, на который нужно ответить: следует ли решать ее своими силами или передать на аутсорсинг? Иными словами, следует ли разрабатывать собственное решение или купить готовое?

В конечном счете это вопрос о приоритетах бизнеса. В общем случае правило гласит: все, что является ключевой компетенцией или конкурентным преимуществом организации, должно оставаться внутри организации, а все неосновное, типовое или рутинное, можно передать для реализации внешнему исполнителю [20]. Например, подавляющее большинство компаний не производит свои процессоры, потому что их дешевле купить у производителей полупроводников.

В сфере разработки программного обеспечения необходимо принять два важных решения: кто будет его создавать и кто — развертывать. Спектр возможных решений показан на рис. 1.2. На одном полюсе находится специализированное программное обеспечение, которое пишется и запускается внутри компании, на другом — широко используемые облачные сервисы или продукты SaaS, которые внедряются и управляются внешним поставщиком, а вы получаете к ним доступ только через веб-интерфейс или API.

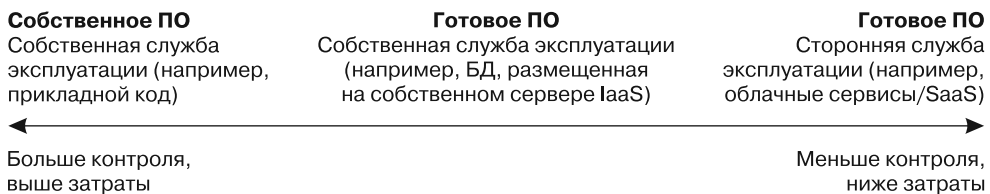


Рис. 1.2. Спектр решений по аутсорсингу программного обеспечения и его функционирования