

ГЛАВА 1

Почему контракты данных нужны именно сейчас

К сожалению, качество данных и его основы — такие, как моделирование данных — были критически понижены в приоритете с появлением больших данных (Big Data), облачных вычислений и современного стека технологий работы с данными. Хотя эти достижения позволили повсеместно задействовать данные в работе разным сотрудникам организаций (в особенности дата-сайентистам и дата-инженерам), простота использования данных также сопровождалась отсутствием ограничений. Это привело к тому, что многие компании накопили значительный технический долг по данным. В условиях, когда дата-команды вынуждены переключаться с исследований и разработок на фактическую генерацию прибыли, а также в связи с переходом от моделицентричного развития искусственного интеллекта (ИИ) к датацентричному организации начали осознавать: качество данных является обязательным условием, а не приятным дополнением. Этому также способствует тесная связь между данными, создаваемыми или обрабатываемыми на более ранних этапах, как правило, разработчиками ПО и подключаемыми на более поздних этапах специализированными дата-командами, которые исторически работали обособленно.

Хотя все это кажется чисто техническими проблемами, в своей основе они сводятся к вопросам управления изменениями в сложных системах. Таким образом, мы считаем, что для масштабирования и поддержки управления изменениями, связанными с данными, в организации необходимы контракты данных — соглашения между производителями и потребителями данных, которые заключаются, обновляются и соблюдаются с помощью API. Прежде чем углубляться в тему контрактов данных и способов их реализации, в этой главе мы поговорим о том, почему специалисты пренебрегли лучшими практиками обеспечения качества данных, почему мы снова уделяем этому приоритетное внимание и какие уникальные условия сложились в индустрии данных после 2020 года — именно они обуславливают необходимость внедрения контрактов для повышения качества данных.

Цикл «мусор на входе — мусор на выходе»

Поговорите с любым специалистом по работе с данными, и он с жаром произнесет мантру «мусор на входе — мусор на выходе» (*garbage in, garbage out; GIGO*) как основную причину большинства проблем и ограничений, связанных с данными. Несмотря на общее понимание сути проблемы на протяжении всего жизненного цикла данных, нам по-прежнему сложно создавать и использовать качественные данные. Во многом это связано с тем, что большинство дата-команд не контролируют источник входных данных, а сами данные часто создаются и обрабатываются для других целей (например, CRUD-операции¹ для программного приложения). Даже если данные хорошо подготовлены для их основного назначения, их часто приходится существенно «причесывать» для вторичного использования, например в аналитике или ML-моделях (машинное обучение). В таких условиях даже самые сильные команды ограничены качеством получаемых данных.

Современное управление данными

Со стороны управление данными кажется относительно простым: данные собираются из исходной системы, проходят через цепочку технологий обработки, образуя то, что мы называем пайплайном, или конвейером, данных, и в итоге попадают на дашборд, который используется для принятия бизнес-решений, или в дата-продукт, например пользовательскую ML-модель. Такое впечатление легко объяснить. Потребители данных, такие как аналитики, менеджеры по продукту и бизнес-руководители, редко видят всю инфраструктуру, отвечающую за транспортировку, очистку, проверку, поиск и преобразование данных, а также создание моделей данных. Подобно масштабной логистической операции, стоимость и объем инфраструктуры для управления потоками данных в нужных бизнес-подразделениях практически незаметны — все работает в фоновом режиме.

По крайней мере, так должно быть. С развитием машинного обучения и искусственного интеллекта данные все чаще оказываются в центре внимания, но организациям по-прежнему сложно извлекать из них реальную ценность. В компаниях, которые испытывают трудности, конвейеры для управления потоками данных начинают давать сбои, дата-сайентисты, нанятые для разработки и внедрения ML-моделей, не могут двигаться дальше, пока не будут решены проблемы с качеством данных, а руководители принимают многомиллионные «решения на основе данных», которые в итоге оказываются ошибочными.

¹ CRUD — акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (*create*), чтение (*read*), модификация (*update*), удаление (*delete*). — *Примеч. науч. ред.*

По мере того как мир продолжает переходить на использование облачных технологий, прежде остававшаяся в тени инфраструктура данных выступает на передний план. Она едва справляется с резко возросшей нагрузкой с точки зрения как объема данных, так и организационной сложности. Именно в тот момент, когда данные готовы выйти на пик своей операционной ценности, наша инфраструктура находится в наихудшем состоянии для достижения этой цели.

Дата-команда оказывается в замешательстве. Подразделения дата-инженеров завалены тикетами (<https://oreil.ly/V99Ie>), поскольку конвейеры данных регулярно выходят из строя по всей компании. Помимо прочего, скрытые сбои приводят к тому, что данные изменяются несовместимым с предыдущими версиями образом, и никто этого не замечает, что приводит к многомиллионным убыткам (<https://oreil.ly/I8h5g>) и, что еще хуже, к потере доверия к данным со стороны как сотрудников, так и клиентов. И это без учета высоких затрат на повторный запуск конвейеров данных и восстановление исторических данных (бэкфил), а также оплату труда высококвалифицированных технических специалистов. Дата-инженеры часто оказываются меж двух огней: с одной стороны — потребители данных, которые не понимают, почему важные для них данные сегодня выглядят иначе, чем вчера, а с другой — производители данных, которые в конечном счете несут ответственность за эти изменения, но не имеют представления о том, кто и как использует их данные. Бизнес часто не до конца понимает роль дата-инженеров. Должны ли они устранять любые проблемы с качеством данных, даже те, которые возникли не по их вине? Несут ли они ответственность за стремительный рост расходов на облачные вычисления в аналитической базе данных (БД)? Если нет, то кто тогда? Опять же, проблемы с управлением изменениями и соблюдением обязательных требований проявляются внутри технических команд.

Такое положение дел является следствием накопившегося долга по данным¹. Под долгом по данным понимается результат компромиссных технологических решений, принятых ради ускорения создания таких дата-активов, как конвейеры данных, дашборды или обучающие наборы данных (датасеты) для ML-моделей. Именно долг по данным выступает главным «антагонистом» книги. Он мешает нам внедрять дата-активы, подрывает доверие к данным и делает итеративный подход к управлению данными практически невозможным. Проблемы команд дата-инженеров возникают из-за долга по данным — либо из-за необходимости напрямую управлять им, либо из-за устранения его каскадных последствий для других команд. В следующих главах вы узнаете, что приводит к долгу по данным, почему с ним сложнее справиться, чем с техническим долгом, и как он в конечном счете может парализовать работу с данными в организации.

¹ Это часть технического долга, которая напрямую относится к данным. — *Примеч. науч. ред.*

Что такое долг по данным

Если вам доводилось работать в крупной ИТ-компании, вы наверняка не раз слышали, как обеспокоенные разработчики повторяли слова «технический долг», вытирая пот со лба и обсуждая будущее, в котором количество запросов к их приложению увеличится в сотни раз.

Проще говоря, технический долг — это результат временных решений, принятых для ускорения развертывания кода в ущерб долгосрочной стабильности. Представьте себе команду разработчиков, которая работает над веб-приложением для компании. У них сжатые сроки для выпуска новой фичи, поэтому они решают пойти по короткому пути и быстро реализовать ее без рефакторинга существующего кода. Быстрая реализация работает, и они укладываются в срок, но это не самое эффективное и удобное в обслуживании решение.

Со временем команда начинает сталкиваться с проблемами. Код новой фичи тесно связан с существующей кодовой базой, что затрудняет добавление или изменение других фич, не вызывая непредвиденных побочных эффектов. Ошибки, связанные с новой добавленной функциональностью, продолжают появляться, и каждый раз, когда разработчики пытаются внести изменения, это занимает больше времени, чем ожидалось, из-за отсутствия надлежащей документации. Затраты на исправление первоначальной реализации с помощью чего-то более масштабируемого — это *долг*.

В какой-то момент этот долг придется «выплатить», иначе команда разработчиков начнет страдать из-за снижения скорости разработки до минимума.

Если вы читаете эту книгу, то, вероятно, сталкивались с подобными проявлениями долга по данным.

- Ключевая метрика, основанная на неверно интерпретированном значении данных, например ошибочном объединении полей `customers` и `active_customers` для расчета выручки.
- Логи для тестирования отправляются («пушатся» — от англ. push) напрямую в базу данных (БД), которая используется для обучения ML-модели, что приводит к неверным результатам прогноза.
- Реализованная на уровне кода логика преобразования данных (например, `CASE WHEN` в SQL), которая изначально решала проблему с качеством данных, больше не соответствует заложенным в данных допущениям.

Как и технический долг, долг по данным является результатом временных решений, принятых ради ускорения работы. Однако долг по данным по нескольким причинам гораздо *серьезнее*, чем технический долг.

Во-первых, в разработке ПО типичный компромисс, приводящий к техническому долгу, — это скорость в ущерб удобству сопровождения и масштабируемости. Другими словами, страдает то, насколько легко разработчикам работать с этой кодовой базой и сколько клиентов или запросов система способна обслужить. При этом работоспособность приложения сохраняется — оно продолжает решать основную задачу пользователя. Однако с данными все иначе: их главная ценность в надежности. Если данным, которые представлены в наших дашбордах, в результатах моделей машинного обучения и программных приложениях, нельзя доверять, то они бесполезны. Чем ниже доверие к данным, тем меньше их практическая ценность. Долг по данным напрямую влияет на их *достоверность*. Быстрое построение конвейеров данных без базовых компонентов надежной инфраструктуры, таких как моделирование данных, документирование и семантическая валидация, напрямую влияет на основную ценность самих данных. По мере накопления долга данные становятся все менее надежными.

Вернемся к нашему примеру с корпоративным веб-приложением. Представьте, что не только поспешная реализация усложнила поддержку кода, но и каждая новая фича делала продукт все менее удобным в использовании, пока от него не отказались все пользователи. Это и есть следствие долга по данным.

Во-вторых, долг по данным гораздо сложнее устранить, чем технический. В большинстве современных компаний, занимающихся разработкой ПО, команды перешли или переходят на микросервисы. Микросервисы — это архитектурный подход, при котором приложение разбивается на набор независимых и слабо связанных компонентов (сервисов), взаимодействующих через облегченные протоколы. Его основная цель — обеспечить возможность разработки и развертывания компонентов системы отдельными командами без зависимости от других команд. Минимизируя взаимозависимости в кодовой базе, разработчики могут развивать свои сервисы с минимальными ограничениями.

В результате программные продукты легче масштабируются и интегрируются с «коробочными» решениями, а организации выстраивают работу своих команд разработчиков вокруг ответственности за конкретные сервисы. Микросервисная архитектура позволяет локализовать технический долг и решать его с конкретной командой. Технический долг, затрагивающий один сервис, не обязательно влияет на другие в той же степени, и это позволяет каждой команде управлять собственным бэклогом без оглядки на задачи по масштабированию всей системы.

Экосистема данных основана на наборе сущностей, представляющих собой распространенные бизнес-объекты. Эти бизнес-объекты соответствуют важным реальным объектам физического мира или концептуальным доменам бизнеса. Например, компания, занимающаяся грузоперевозками, может

использовать такие сущности, как отправления, грузоотправители, перевозчики, грузовики, клиенты, счета-фактуры, договоры, несчастные случаи и объекты. Сущности — это «существительные» языка данных, они являются строительными блоками для вопросов, на которые в конечном счете можно ответить с помощью запросов.

Однако большинство ценных данных в организации проходят через ряд преобразований, выполняемых дата-инженерами, аналитиками и инженерами-аналитиками. Преобразования объединяют реальные данные на уровне предметной области в логически сгруппированные агрегаты — факты, которые используются в метриках для оценки состояния бизнеса. Например, метрика «отток клиентов» может объединять данные о клиенте и платежах, а «количество завершенных поставок в пункты назначения» — данные о перевозках и объекте назначения. Поскольку создание метрик может занимать много времени, большинство запросов зависят как от основных бизнес-объектов, так и от агрегированных данных. Это означает, что команды, работающие с данными, тесно связаны друг с другом и с артефактами, которые они создают, что является существенным отличием от микросервисов.

Такая тесная связь означает, что накопленный долг по данным нельзя легко устранить изолированно. Даже небольшое изменение одного запроса может серьезно повлиять на всю цепочку данных, радикально меняя отчеты и даже пользовательские продукты данных. Практически невозможно узнать, где и как используются данные, а также насколько важен тот или иной дата-актив для бизнеса. Чем больше накапливается долг между производителями и потребителями данных, тем сложнее распутать паутину запросов, фильтров и плохо построенных моделей данных, что ограничивает прозрачность на последующих этапах обработки данных.

Подводя итог: долг по данным — это порочный круг. Он подрывает доверие к данным, что снижает базовую ценность, которую должны нести данные. Поскольку команды, работающие с данными, тесно связаны друг с другом, устранить долг без каскадирования последствий для всей организации невозможно. По мере его роста экспоненциально снижается и уровень доверия к данным, и в конечном счете это затрагивает почти все бизнес-домены и приводит к организационному хаосу. Долг по данным — самая серьезная проблема, которую предстоит решить относительно данных, и это еще мягко сказано.

Как работает принцип «мусор на входе — мусор на выходе»

Проблема долга по данным актуальна практически для всех отраслей и вертикалей бизнеса. На первый взгляд может показаться, что управление таким долгом — это обычное дело для дата-команд: неизбежная судьба, которая ждет любую организацию, работающую с данными, даже если в компании к ним относятся серьезно. Однако существует одна категория компаний, которая

редко сталкивается с проблемой долга по данным, и причина этого может вас удивить: стартапы.

Когда мы говорим «стартап», то имеем в виду компанию на ранней стадии развития в прямом смысле этого слова: около 20 разработчиков или меньше, небольшая, но эффективная дата-команда (пусть даже это всего один-два дата-инженера и несколько аналитиков), а также продукт, который либо уже соответствует потребностям рынка, либо находится на пути к этому. В ходе подготовки этой книги мы пообщались с десятками компаний такого профиля, и почти все они сообщают не только о минимальном долге по данным, но и о практически полном отсутствии проблем с качеством данных. Причина этого проста: чем меньше инженерная команда, тем проще наладить коммуникацию в случае изменений.

В большинстве крупных компаний сложная управленческая иерархия с большим количеством инженерных и дата-команд, которые редко взаимодействуют друг с другом. Например, в компании Convoу инженерные команды были разделены на «отряды» (pods), более известные как сквады (squads) — термин, заимствованный из продуктовой модели управления компании Spotify. Сквады — это небольшие команды, созданные для решения основных клиентских проблем или задач бизнес-доменов, которые обеспечивают максимальную гибкость, независимость в принятии решений и адаптивность. Один сквад занимался поддержкой нашей операционной команды, создавая ML-модели для определения приоритетности наиболее важных инцидентов, о которых сообщали клиенты. Другой работал над моделью ценообразования компании Convoу, а третий мог сосредоточиться на предоставлении нашим крупнейшим контрагентам аналитических данных в режиме реального времени о предполагаемой дате доставки.

Хотя все эти команды входили в состав одного крупного подразделения, их дорожные карты составлялись менеджерами по продукту обособленно друг от друга. Менеджеры по продукту редко общались с другими командами, если только им что-то не было нужно напрямую. Это привело к возникновению серьезных проблем у дата-команд, когда были выпущены новые фичи продукта. Разработчик, отвечающий за БД, может принять решение переименовать столбец, удалить его, скорректировать бизнес-логику, прекратить генерацию события или решить любую другую проблему, которая создаст трудности для потребителей данных на последующих этапах. Потребители данных часто первыми замечают изменения, потому что что-то не так отображается на их дашборде или ML-модель начинает выдавать некорректные прогнозы.

В небольших стартапах дата-инженеры и другие технические специалисты по работе с данными еще не разделились на несколько обособленных команд с разными стратегиями. Все они входят в команду с единой стратегией.

Специалисты по данным знают практически обо всех вносимых изменениях и могут легко поднять руку на собрании или отвести ведущего инженера в сторону, чтобы объяснить проблему. Однако в быстро развивающейся компании с десятками, сотнями или тысячами инженеров это уже невозможно. Такой разрыв в коммуникации приводит к наиболее часто упоминаемому феномену, связанному с качеством данных: мусор на входе — мусор на выходе (GIGO).

GIGO возникает, когда в конвейер данных попадают данные, не соответствующие ожиданиям бизнес-заказчиков. Проблема GIGO заключается в том, что с ней можно справиться только *постфактум*, то есть решение проблемы всегда будет сопряжено с определенными затратами. В некоторых случаях эти издержки могут быть существенными — например, потеря выручки из-за того, что руководитель принял неверное решение на основе дашборда с искаженными данными, которые за сутки существенно изменились (например, стали неверными, но выглядели правдоподобно), или из-за того, что ML-модель начала давать неверные прогнозы о покупательском поведении. В других случаях последствия могут быть менее серьезными: дашборд демонстрирует неверные показатели, которые можно легко исправить перед следующей презентацией с помощью простого оператора CASE. Однако простые и быстрые исправления скрывают за собой более серьезную проблему, которая назревает подспудно, — рост долга по данным.

По мере увеличения количества ретроспективных исправлений в критически важных узлах экосистемы данных начинают формироваться «горячие точки» концентрации экспертных (зачастую недокументированных) знаний. Массивные файлы SQL, в которых строка за строкой записана бизнес-логика, совершенно непонятны никому, кроме самых первых специалистов по работе с данными в компании. Неясно, что означают эти данные, кто их владелец, откуда они берутся и почему те или иные данные приводят к неожиданным результатам без каких-либо пояснений в документации.

Со временем долг по данным, вызванный GIGO, начинает расти в геометрической прогрессии по мере увеличения соотношения разработчиков ПО и специалистов по данным. Количество деплоев кода увеличивается с нескольких в неделю до сотен или тысяч в день. Критические изменения становятся обычным явлением, в то время как многие проблемы с качеством данных, влияющие на их содержание (бизнес-логику), могут оставаться незамеченными днями, неделями или даже месяцами. Когда проблема становится настолько серьезной, что начинает заметно замедлять работу аналитиков и дата-сайентистов, вы уже достигли критической точки: без радикальных мер выхода практически нет. Долг по данным будет расти, а пользовательский опыт — ухудшаться. Дата-инженеры и дата-сайентисты начнут увольняться в поисках менее стрессовых условий труда, а ценность самого значимого актива компании — данных — будет снижаться.

Хотя GIGO является наиболее распространенной причиной долга по данным, проблемы, связанные с данными, кроются и в общепринятых архитектурных подходах, которые мы используем.

«Смерть» хранилищ данных

Начиная с конца 1980-х годов и по сей день хранилище данных (data warehouse) остается ключевым компонентом практически любой экосистемы данных и основой практически всех аналитических сред.

Хранилище данных — одно из наиболее часто упоминаемых понятий в сфере данных, и его важно понимать на базовом уровне, если мы хотим разобраться в причинах стремительного роста долга по данным. Билл Инмон по праву считается «отцом хранилища данных»: именно он сформулировал эту концепцию. По словам Инмона, «хранилище данных — это предметно-ориентированная, интегрированная, зависящая от времени¹ и неизменяемая² совокупность данных, служащая для поддержки процесса принятия управленческих решений».

По мнению Инмона, хранилище данных — это не просто репозиторий, а предметно-ориентированная структура, которая соответствует тому, как организация воспринимает свои данные с семантической точки зрения. Эта структура обеспечивает целостное представление о бизнесе, позволяя лицам, принимающим решения, глубже понять тенденции и закономерности, а также использовать данные для визуализации, машинного обучения и решения операционных задач.

Чтобы структура данных считалась хранилищем, она должна обладать тремя основными свойствами.

- Хранилище данных проектируется вокруг таких ключевых бизнес-сущностей компании, как клиенты, продукты и продажи, а также других, связанных с конкретной предметной областью.
- Данные в хранилище поступают из различных систем-источников компании. Данные объединяются в общий формат, что позволяет устранить несогласованность и дублирование. Такая интеграция создает *единый источник правды* и позволяет потребителям данных надежно связываться с системами-источниками, не беспокоясь о процессах репликации данных.
- Данные в хранилище накапливаются и хранятся с течением времени, что позволяет проводить анализ исторических данных. Это важно для анали-

¹ В хранилище данных аккумулируется информация за продолжительные интервалы времени, значительно большие, чем в операционных системах. — *Примеч. пер.*

² Данные доступны только для чтения. — *Примеч. пер.*

тики, привязанной ко времени, например для понимания того, сколько клиентов приобрели товар за 30 дней, или для отслеживания трендов в данных, которые можно использовать в машинном обучении или других видах предиктивной аналитики. В отличие от операционных БД, которые постоянно обновляются по мере появления новых транзакций, хранилище данных остается неизменяемым: однажды загруженные данные не изменяются, обеспечивая стабильную среду для анализа.

Построение хранилища данных обычно начинается с создания диаграммы связи сущностей (entity relationship diagram, ERD), как показано на рис. 1.1. ER-диаграмма представляет собой логическую и семантическую структуру основных операций компании и служит своего рода картой, которой можно руководствоваться при разработке хранилища данных. Сущность — это бизнес-объект (предметная область), который можно представить в виде таблицы, где каждая строка соответствует уникальному экземпляру этого объекта. Каждой сущности соответствует набор измерений, которые содержат конкретные сведения о сущности в виде столбцов. Например, сущность «клиент» (не связанная с приведенным ниже примером ER-диаграммы) может содержать такие измерения, как:

- `Customer_id` — идентификатор клиента;
- `Birthday` — день рождения;
- `FirstName` — имя;
- `LastName` — фамилия.

Внешние ключи — важный компонент при проектировании ER-диаграмм. Это уникальные идентификаторы, которые позволяют аналитикам объединять данные из нескольких сущностей в одном запросе. Например, таблица `customers_table` может содержать следующие внешние ключи:

- `Address_id` — идентификатор адреса;
- `Account_id` — идентификатор аккаунта.

Используя внешние ключи, дата-сайентист легко определит количество авторизаций для каждого пользователя или количество регистраций на сайте по городам или регионам.

Связь между двумя сущностями называется *кардинальностью*. Именно она позволяет аналитикам понять природу отношений между сущностями, что крайне важно для обеспечения достоверной аналитики и ее масштабирования. Например, если у объекта «клиент» связь «один к одному» с объектом «аккаунты», то мы не ожидаем увидеть более одного аккаунта, привязанного к пользователю или адресу электронной почты.

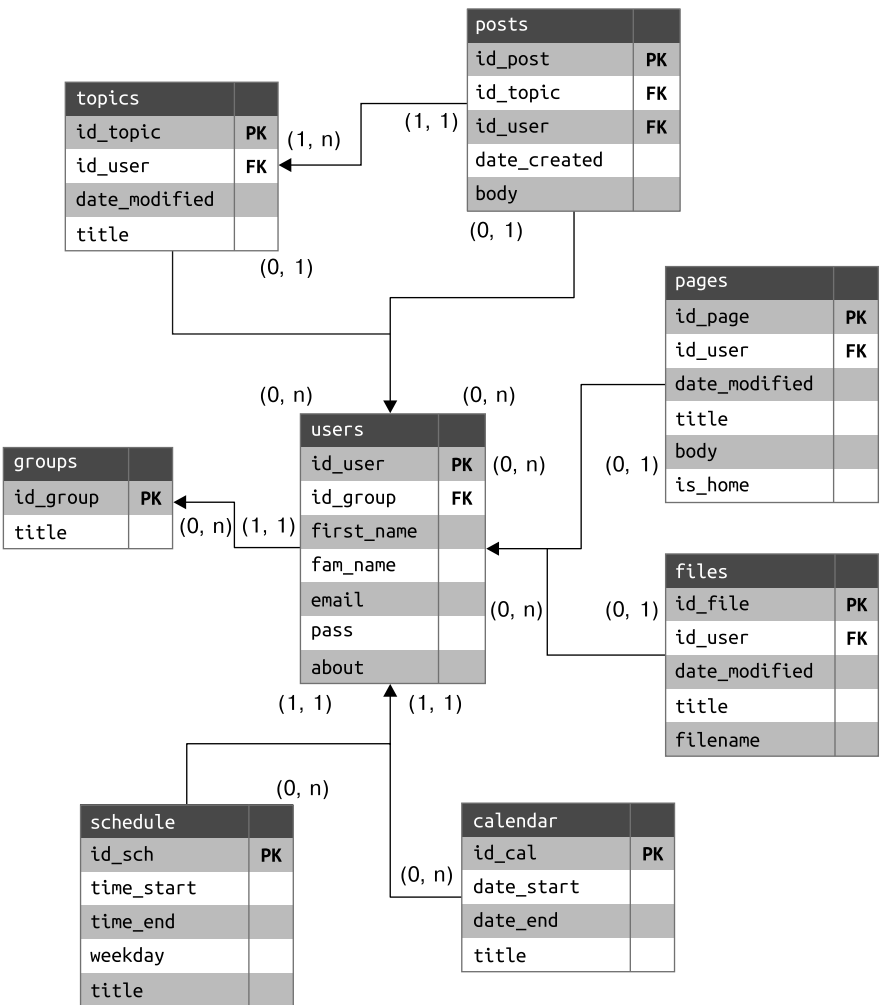


Рис. 1.1. Пример диаграммы «сущность — связь» (<https://oreil.ly/3-SAp>)

Такие сопоставления (мэппинги) невозможно провести только на основе интуиции. На самом высоком уровне абстракции домены данных и их роль в бизнесе описываются *концептуальной моделью данных*. Затем начинается процесс определения идеального набора сущностей и их измерений, внешних ключей и кардинальности. Этот процесс называется построением *логической модели данных*, а результат преобразования семантической карты в таблицы, столбцы и индексы, к которым можно обращаться с помощью аналитических языков, таких как SQL, — *физической моделью данных*. Три процесса вместе представляют собой процесс моделирования данных. Только благодаря тщательному моделированию можно создать полноценное хранилище данных.

Первоначальное значение терминов «хранилище данных» и «моделирование данных» позволяет понять, почему современная экосистема данных находится в таком плачевном состоянии. Но прежде, чем мы перейдем к современной эпохе, стоит поговорить о том, как использовались хранилища данных в период их расцвета.

Предшествующая эпоха

Хранилища данных появились еще до того, как стали популярными облачные технологии, программное обеспечение и даже Интернет. В эпоху до появления Интернета создание хранилища данных требовало специализированной программно-аппаратной реализации во внутренней сети организации. Компаниям требовались выделенное серверное оборудование и системы хранения из-за больших объемов данных, для обработки которых предназначались хранилища. Для развертывания хранилища данных использовались высокопроизводительные вычислительные серверы, а сами данные управлялись с помощью систем управления реляционными базами данных (РСУБД), таких как Oracle, IBM DB2 или Microsoft SQL Server.

Задачи, ради которых создавались хранилища данных, как правило, носили операционный характер. Ретейлеры могли анализировать покупательское поведение, сезонные колебания потока посетителей и покупательские предпочтения в разных торговых точках, чтобы более эффективно управлять товарными запасами. Производители могли выявлять узкие места в своих цепочках поставок и отслеживать производственные графики. Компания Ford сэкономила более миллиарда долларов, применяя хранилище данных и совершенствуя процессы на основе принципов *кайдзен* (Kaizen), чтобы оптимизировать операционную деятельность за счет более качественных данных. Авиакомпании использовали хранилища данных для планирования оптимальных маршрутов, времени вылета и численности экипажа.

Однако создание хранилищ данных было недешевым процессом. Требовалось нанимать специалистов для проектирования, реализации и управления внедрением ER-диаграмм, моделей данных и в конечном счете самого хранилища. Командам разработчиков нужно было работать в тесном контакте с профильным руководством, чтобы координировать взаимодействие между транзакционными и аналитическими системами (OLTP и OLAP соответственно). Для внедрения и эксплуатации дорогостоящих инструментов ETL, таких как Informatica, Microsoft SQL Server Integration Services (SSIS), Talend и других, требовались эксперты. В целом переход к функционирующему хранилищу данных может занять годы, потребовать миллионы долларов вложений и участия десятков специализированных сотрудников.

Руководителями этого процесса были архитекторы данных, инженеры широкого профиля с опытом работы в сфере компьютерных наук и специализацией

в области управления данными. Архитектор должен был разработать исходную модель данных, составить план внедрения, закупить и внедрить инструменты, довести план до сведения бизнес-заказчиков, а также координировать управление и постоянное совершенствование хранилища с течением времени. Они выступали в роли дата-стюардов, следя за тем, чтобы пользователи всегда получали актуальные и достоверные данные, соответствующие их потребностям. Какое-то время это была лучшая в своем классе модель, но потом все начало меняться.

Программное обеспечение поглощает мир

В 2011 году Марк Андрессен, венчурный инвестор и соучредитель легендарной венчурной компании Andreessen Horowitz, написал эссе под названием «Почему программное обеспечение поглощает мир» (*Why Software Is Eating the World*) для *The Wall Street Journal*. В нем он объяснил, какое стремительное и трансформирующее влияние оказывают ПО и технологии на различные отрасли:

Инструменты разработки и интернет-сервисы упрощают запуск новых ИТ-стартапов по всему миру во многих отраслях — без необходимости инвестировать в новую инфраструктуру и обучать новых сотрудников. В 2000 году, когда мой партнер Бен Хоровиц был генеральным директором первой компании, занимавшейся облачными вычислениями, — Loudcloud — стоимость запуска базового интернет-приложения для клиента составляла примерно 150 000 долларов в месяц. Сегодня запуск такого приложения в облаке Amazon стоит около 1500 долларов в месяц.

Двухтысячные годы ознаменовались невероятными изменениями в сфере бизнеса. После «пузыря доткомов» в конце 1990-х годов на мировой арене появились новые технологические гиганты в виде высокодоходных и низкочередных стартапов с ошеломляющими темпами технологических инноваций и роста. Сергей Брин и Ларри Пейдж превратили Google из поисковой системы, работавшей из комнаты общежития Стэнфордского университета в 1998 году, в глобальный рекламный гигант с рыночной капитализацией более 23 миллиардов долларов к концу 2000-х годов. Amazon практически вытеснила Walmart с позиции лидера в сфере торговли, Netflix убила Blockbuster, а Facebook превратилась из студенческого приложения для поиска друзей в компанию с годовым доходом 153 миллиона долларов (<https://oreil.ly/xsCeU>) всего за три года.

Одним из важнейших внутренних изменений, вызванных ростом числа ИТ-компаний, стало распространение Agile — методологии гибкой разработки ПО, популяризированной консалтинговой компанией Thoughtworks. До этого момента релизы осуществлялись последовательно: команды проектировали, разрабатывали, тестировали и развертывали продукт целиком.

Такая каскадная модель (Waterfall) напоминала выпуск фильмов, когда клиент получает готовый продукт, который был тщательно проверен и прошел строгий контроль качества. Однако методология Agile предложила иной путь. Вместо того чтобы ждать, когда продукт будет полностью готов к выводу на рынок, компании выпускали релизы более итеративно, уделяя особое внимание быстрому управлению изменениями и коротким циклам обратной связи с пользователями.

Такие компании, как Google, Facebook и Amazon, одними из первых внедрили Agile. Марк Цукерберг однажды сказал, что философия разработки Facebook заключается в том, чтобы «двигаться быстро и ломать стереотипы». Высокая скорость внедрения позволяла Agile-компаниям быстро переориентировать свою деятельность на то, за что клиенты с наибольшей вероятностью готовы платить. Синхронизация потребностей клиента и характеристик продукта привела к своего рода нирване, которую венчурные инвесторы называют соответствием продукта требованиям рынка. То, на что у традиционных компаний уходило десятилетия, интернет-компании достигали за считанные годы.

С появлением Agile в качестве основного подхода для компаний, ориентированных на разработку ПО, начала формироваться соответствующая организационная структура. Разработчики стали центральной фигурой в R&D-подразделениях, которые для лаконичности и точности переименовали в продуктовые. Стали появляться новые роли, предполагающие поддержку разработчика: UX-дизайнеры, специализирующиеся на веб-дизайне и дизайне приложений; менеджеры по продукту, которые совмещают управление проектами со стратегией продукта и позиционированием бизнеса, чтобы помочь в создании дорожной карты; аналитики, которые собирают логи приложения, чтобы рассчитать ключевые метрики, такие как конверсия в регистрацию, отток пользователей и продуктовая активность. Команды стали меньше и более узкопрофильными, что позволило разработчикам выкатывать код еще быстрее, чем раньше.

По мере роста технического разрыва традиционные офлайн-компании начали ощущать давление со стороны рынка и инвесторов, которые требовали от них перехода к модели Agile. Скорость роста технологических компаний вызывала тревогу (рис. 1.2), и существовал риск, что компании, перешедшие на подобную бизнес-модель, получают слишком большое преимущество, чтобы их можно было догнать. Примерно в 2015 году термин «цифровая трансформация» начал стремительно набирать популярность. Консалтинговые компании, такие как McKinsey, Deloitte и Accenture, продвигали услуги по модернизации технической инфраструктуры традиционных компаний путем создания приложений и сайтов и, что наиболее важно в контексте книги, перехода от локальных баз данных к облачным.



Первый вариант этого раздела был написан в 2023 году — еще до резкого скачка в 2025 году, который виден на графике (рис. 1.2). Хотя облачные технологии сыграли важную роль в резком росте с 2013 по 2025 год, было бы упущением не отметить сильную корреляцию между почти вертикальной линией и бумом ИИ в 2025 году.

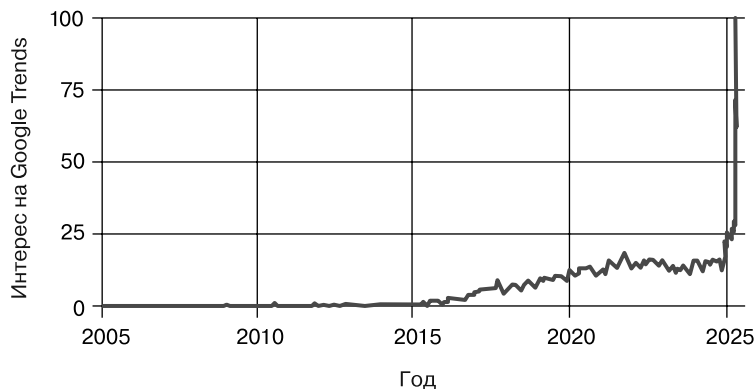


Рис. 1.2. Анализ данных Google Trends по поисковому запросу «цифровая трансформация»

Самым большим преимуществом облачных технологий были экономия средств и повышение скорости. Такие компании, как Amazon (AWS), Microsoft (Azure) и Google (GCP), избавили своих клиентов от необходимости обслуживать физические серверы, что позволило сэкономить миллионы долларов на инфраструктуре и человеческих ресурсах. Это соответствовало парадигме Agile, которая позволяла компаниям работать намного быстрее, перекладывая сложные задачи на поставщика услуг. Такие компании, как McDonald's, Coca-Cola, Unilever, General Electric, Capital One, Disney и Delta Airlines — примеры крупных корпораций из списка *Fortune 500*, которые вложили значительные средства в цифровую трансформацию и переход к использованию облачных технологий.

Переход к микросервисам

В начале 2000-х годов наиболее распространенным архитектурным подходом в разработке ПО был монолит. *Монолитная архитектура* — это традиционный вариант проектирования приложений, при котором система создается как единое целое с тесно интегрированными компонентами и одной БД. Все модули являются частью одной кодовой базы, и приложение развертывается целиком. Из-за высокой связанности компонентов разработчикам сложно поддерживать монолиты по мере их роста, что приводит к значительному снижению скорости выпуска релизов. В 2011 году (в том же году, когда Марк

Андриссен опубликовал свою культовую статью в *The Wall Street Journal*) появилась новая парадигма — микросервисы.

Микросервисы — это архитектурный подход, который изменяет структуру приложений, разбивая их на набор независимых и слабо связанных компонентов (сервисов), которые взаимодействуют с помощью облегченных протоколов. Главная цель такого подхода — обеспечить разработку и развертывание сервисов отдельными командами независимо друг от друга. Сводя к минимуму взаимозависимости в кодовой базе, разработчики могут развивать отдельные компоненты с минимальными ограничениями. В результате программные продукты легче масштабируются и интегрируются с «коробочными» решениями, а организации выстраивают работу своих команд разработчиков вокруг ответственности за конкретные сервисы. Термин «микросервис» был впервые введен консультантами Thoughtworks и получил широкое распространение благодаря публикациям Мартина Фаулера (<https://martinfowler.com>).

Компании-разработчики ПО с энтузиазмом перешли на микросервисы, чтобы еще больше разделить компоненты своей инфраструктуры и повысить скорость разработки. Такие корпорации, как Amazon, Netflix, Twitter (с 2023 года — X) и Uber, были одними из самых быстрорастущих компаний, внедривших данный архитектурный подход, и это сразу же сказалось на их масштабируемости. Вернер Фогельс, технический директор Amazon, говорил:

«Наши сервисы построены на основе микросервисов. Архитектура на основе микросервисов, в которой программные компоненты разделены и могут развертываться независимо друг от друга, легко адаптируется к изменениям, обладает высокой масштабируемостью и отказоустойчивостью. Она позволяет непрерывно развертывать приложения и часто экспериментировать с ними».

Архитектура данных в плачевном состоянии

В стремительно развивающемся мире микросервисов, облачных технологий и Agile была одна молчаливая «жертва» — архитектор данных. Изначально архитекторы данных были своего рода распорядителями, которые создавали ER-диаграммы, контролировали доступ к потокам данных, проектировали OLTP- и OLAP-системы и выступали в роли поставщика централизованного источника достоверных данных. Однако в новом мире децентрализации, обособленных команд разработки и высокой скорости разработки архитектор данных воспринимался (вполне обоснованно) как фактор, замедляющий развитие.

На внедрение функциональной архитектуры данных уходило слишком много времени. Тратить месяцы на создание идеальной ER-диаграммы было слишком долго. Архитекторы данных утратили контроль. Они больше не могли диктовать разработчикам, как проектировать базы данных, а без централизованной

модели данных, на которую можно было бы опираться, концептуальные и физические модели данных перестали синхронизироваться. Данные должны были передаваться быстро, а продуктовые команды не хотели зависеть от третьей стороны, которая предоставила бы тщательно отобранные данные в готовом виде. Продуктовые команды были больше заинтересованы в создании минимально жизнеспособных продуктов (MVP), экспериментах и итерациях в поиске наилучших решений. Идеальное качество данных на начальном этапе уже не считалось необходимым.

Чтобы можно было обеспечить быстрый доступ к данным с первого дня, появился архитектурный паттерн *озера данных*. Озеро данных — это централизованное хранилище, в котором структурированные, неструктурированные и полуструктурированные данные могут храниться практически в любом объеме с минимальными затратами. Озера данных обычно размещаются в высокомасштабируемых системах хранения данных, например Amazon S3, Azure Data Lake Storage и Google Cloud Storage. Команды аналитиков и дата-сайентистов могут либо извлекать данные из озера данных, либо анализировать/запрашивать их напрямую с помощью фреймворков вроде Apache Spark или поставщиков SaaS-решений, таких как Looker или Tableau.

Хотя озера данных были эффективны в краткосрочной перспективе, отсутствие контроля приводило к серьезным проблемам с качеством данных из-за отсутствия четко определенных схем данных, требуемых для функционирования баз данных OLAP. В конечном счете это мешало организациям использовать данные в более структурированном виде. Появились аналитические БД, такие как Redshift, Big Query и Snowflake, благодаря которым дата-команды могли проводить сложный анализ в течение более длительных периодов времени. Данные извлекались из исходных систем и помещались в озеро данных и аналитические среды, чтобы дата-команды имели доступ к свежим данным в любое время.

Бизнес пришел к выводу, что им нужны *дата-инженеры*, создающие конвейеры для перемещения данных между OLTP-системами, озерами данных и аналитическими средами, а не архитекторы данных с их более жесткими и негибкими подходами к проектированию.

Упразднение должности архитектора данных привело к постепенному отказу от хранилищ данных. Во многих компаниях хранилища данных существуют только на бумаге. Нет целостной модели данных, нет четко определенных сущностей, а существующая структура уже не выполняет роль полноценного интеграционного слоя, отражающего бизнес-процессы в коде. Сегодня дата-инженеры делают все возможное, чтобы определить общие бизнес-сущности в своей аналитической среде, но на них оказывается давление с обеих сторон — со стороны потребителей и производителей данных. Первые всегда стремятся работать быстрее и «ломать систему», а вторые действуют изолированно,

редко задумываясь о том, куда направляются создаваемые ими данные и как они используются.

Скорее всего, джинна уже не загнать обратно в бутылку. Гибкие методы разработки и микросервисная архитектура приносят реальную пользу и позволяют компаниям экспериментировать быстрее и эффективнее, чем их конкуренты. Сейчас требуется эволюция как роли архитектора данных, так и самого хранилища данных. Нам нужно решение, созданное для современных условий и способное приносить постепенную, но ощутимую пользу там, где это действительно важно.

Развитие современного стека данных

Термин «современный стек данных» (Modern Data Stack, MDS) относится к набору инструментов, предназначенных для облачных архитектур данных. Эти инструменты частично совпадают со своими предшественниками из «до-облачной» эпохи, но, как правило, они более модульны и используются для более широкого круга задач. MDS — чрезвычайно важное средство в наборе инструментов стартапа. Поскольку новые компании изначально являются облачными, то есть с самого начала разрабатываются с использованием облачных технологий, для реализации анализа данных или машинного обучения в любом масштабе в конечном счете потребуется внедрение хотя бы части MDS.

Крупные игроки

Самым крупным и популярным решением в рамках MDS считается облачная аналитическая база данных Snowflake — нередко именно с ним связывают появление самого термина. Компания Snowflake вышла на биржу 16 сентября 2020 года. Во время первичного публичного размещения акций (IPO) компания была оценена примерно в 33,3 миллиарда долларов, что стало одним из крупнейших IPO в истории разработки ПО. Облачная архитектура Snowflake, в которой вычисления отделены от хранения данных, обеспечивает значительную масштабируемость, устраняя аппаратные ограничения и необходимость в ручной настройке. Это гарантирует высокую производительность даже при больших нагрузках благодаря динамическому распределению ресурсов для каждого запроса. Snowflake позволяет компаниям, интенсивно использующим вычислительные ресурсы, сэкономить сотни тысяч, а то и миллионы долларов, предоставляя множество возможностей для интеграции с другими дата-продуктами.

Существуют и другие популярные альтернативы Snowflake, такие как Google BigQuery, Amazon Redshift и Databricks. В то время как Snowflake фактически монополизировала рынок преобразования данных с SQL в облаке, Databricks

предлагает наиболее полную среду для унифицированной аналитики, созданную на основе самого популярного в мире фреймворка для распределенной обработки больших данных с открытым исходным кодом — Spark. С помощью Databricks дата-сайентисты и аналитики могут легко управлять своими кластерами Spark, строить визуализации с использованием таких языков, как Python или Scala, обучать и внедрять ML-модели и многое другое. Конкуренция между Snowflake и Databricks фактически переросла в настоящую «гонку вооружений» в сфере данных и привела к появлению новой волны стартапов, ориентированных на работу с данными, с конца 2010-х до начала 2020-х годов.

Однако аналитические БД не единственные компоненты современной инфраструктуры данных. Широкое признание также получил инструмент извлечения, загрузки и преобразования данных (ELT) от компании Fivetran. Несмотря на то что на момент написания книги акции Fivetran не торговались на бирже, компания оказала значительное влияние на современный ландшафт технологий работы с данными. Благодаря набору удобных интерфейсов и готовых коннекторов Fivetran позволяет дата-инженерам подключаться напрямую к источникам и приемникам данных, быстро извлекая и загружая данные из БД, приложений и API. Для многих компаний на ранних стадиях развития Fivetran фактически стал стандартом перемещения данных между источниками и приемниками.

Еще один важный компонент — dbt (сокращение от data build tool), один из самых быстрорастущих проектов с открытым исходным кодом в рамках современного стека данных. Благодаря модульным преобразованиям, реализуемым с помощью Yet Another Markdown Language (YAML), dbt предоставляет интерфейс командной строки, который позволяет дата-инженерам и аналитикам работать с преобразованиями в стиле, близком к разработке ПО. Облачная версия dbt расширяет возможности продукта от простых преобразований до формирования слоя метрик на основе YAML, предоставляющего дата-командам возможность определять и хранить факты и их метаданные, которые могут быть использованы при проведении экспериментов и последующего анализа.

Это лишь часть инструментов в рамках MDS. За последнее время появились десятки компаний и решений, от систем оркестрации, например Airflow и Dagster, до инструментов наблюдаемости данных, таких как Monte Carlo и Anomalo, облачных каталогов данных вроде Atlan и Select Star, репозитория метрик, хранилищ признаков и платформ для экспериментов и т. д.

Причина, по которой в последние годы ускорилось развитие средств для работы с данными, заключается в первую очередь в простоте их интеграции с наиболее популярными аналитическими базами данных: Snowflake, BigQuery, Redshift и Databricks. Все эти инструменты предоставляют удобные API и хорошо структурированные метаданные, которые можно использовать для анализа, написания преобразований или выполнения запросов к данным.