Оглавление

Введение

Вряд ли сегодня кому-то удастся назвать современную область знаний, в которой не нашли бы применения компьютерные базы данных (БД). Наука, образование, экономика, электронная коммерция, медицина, статистика, военное дело – список можно продолжать очень долго. Базы данных сопровождают человека на протяжении всей жизни. Появление на свет ребенка, учеба в школе и вузе, получение паспорта или водительских прав, посещение врача, покупки в магазинах, поиск книги в библиотеке, открытие счета в банке – в современном информационном обществе ни одно из этих и многих других событий не обходится без появления очередной электронной пометки в памяти многочисленных компьютеров.

Базы данных входят в десятку самых востребованных программных продуктов и служат источником неплохого заработка для профессиональных разработчиков. Судите сами, без хранения и учета данных сегодня обойтись весьма сложно. Многочисленные магазины, склады, страховые агентства, отделы кадров, бухгалтерии, учебные заведения и множество других предприятий и организаций остро нуждаются в разработанных специально для них БД. И спрос все еще превышает предложение.

Создать эффективную базу данных весьма непросто, даже если она предназначена для обслуживания незначительных объемов данных и подлежит эксплуатации на домашнем компьютере. Сложность проекта возрастает на порядок, когда возникает задача разработать жизнеспособный коммерческий продукт, с которым смогут одновременно работать десятки пользователей. Именно поэтому главная задача книги – вооружить читателя знаниями о технологии проектирования баз данных. Здесь вы подчерпнете всю необходимую информацию о:

- задачах, решаемых с помощью БД;
- архитектуре систем управления базами данных (СУБД);
- реляционной, объектно-ориентированной, документ-ориентированной (и ряде других) моделях данных;
- жизненном цикле проектов БД, этапах концептуального, логического и физического проектирования БД;
- особенностях физического хранения и правилах индексирования данных;

- многопользовательском доступе к данным и управлении транзакциями;
- особенностях организации доступа к БД и обеспечении безопасности данных;
- структурированном языке запросов SQL;
- применении в SQL регулярных выражений;
- расширяемом языке разметки XML;
- многомерном анализе данных;
- централизованных и распределенных БД;
- правилах разработки клиентских приложений БД;
- особенностях документирования проектов БД.

Самое главное, что получит читатель после изучения предложенного материала, – владение методологией работы с любой БД. Книга не ограничивает читателя в вопросе выбора целевой СУБД, поэтому ваша база данных может быть развернута как на основе простейших настольных систем, так и на фундаменте профессиональных многопользовательских клиент-серверных программных решений.

Соглашения

Для акцентирования внимания читателя на ключевых частях излагаемого материала такой текст отмечен особым форматированием:

> **Замечание**
> Таким способом в тексте книги выделен материал, который вы должны принять к сведению. Обычно это определения, комментарии или замечания.

Кроме того:

- впервые встречающиеся термины и определения выделены **полужирным шрифтом**;
- впервые встречающиеся аббревиатуры комментируются;
- код примеров, синтаксические конструкции даны моноширинным шрифтом;
- помимо содержания, книга включает подробный предметный указатель, позволяющий найти в тексте интересующую читателя информацию.

Глава 1

Эволюция баз данных

Очень сложно привести пример какой-либо области науки и техники, сделавшей за последние полстолетия столь же значимый рывок в своем развитии, как современная микроэлектроника и идущие с ней рука об руку информационные технологии (ИТ). Более того, ИТ развиваются столь динамично, что даже специалистам в этой области приходится едва ли не каждые 5–6 лет кардинально переучиваться, дабы успеть попасть в последний вагон улетающего вдаль с космической скоростью экспресса технологий разработки программного обеспечения.

Как человек сумел за столь короткий срок совершить столь большие шаги в области ИТ? Чтобы ответить на этот вопрос, стоит разобраться с тем, что происходило в те «старозаветные» времена, в которых компьютер упоминался лишь в произведениях писателей-фантастов. Для этого нам предстоит отмотать временную ленту всего на несколько десятилетий назад и попасть в эпоху, когда на смену механическому арифмометру стали приходить первые вычислительные машины. Именно этот момент времени мы и станем считать началом всех начал для современных информационных технологий в целом и зарождающихся баз данных (БД) в частности.

Для полного погружения в эпоху нарисуем картину научно-вычислительной лаборатории 50-х годов XX века. Посреди огромного зала всеми цветами радуги переливается огромный ламповый монстр – электронно-вычислительная машина. А вокруг нее вьется рой инженеров, математиков и программистов. Инженеры меняют радиодетали, математики выдумывают формулы, а программисты замыкают круг – на основе предложенных математиками формул закладывают в ЭВМ программы, чтобы последние жгли электронные лампы. Огромное количество обслуживающего персонала, трудящегося во благо научно-технического прогресса, в некоторой степени роднило обслуживание ЭВМ с ходом

возведения Вавилонской башни (как с точки зрения процесса, так и по результату).

Технология разработки прикладного программного обеспечения тех времен был достойна кисти Сальвадора Дали. Наивный заказчик расчетов приходил к математику и просил, чтобы машина ответила – сколько, на ее взгляд, будет равняться 2 + 2. Математик-алгоритмист в глубокой задумчивости рисовал алгоритм и передавал его программисту, перекладывающему алгоритм на низкоуровневый язык машинных команд. Для того чтобы ЭВМ смогла усвоить программу, последняя набивалась техниками на перфоленту или на перфокарты. Данные полдня загружались в память машины, затем она пару-другую раз зависала, шипела, кряхтела и, наконец, к всеобщей радости, выдавала распечатку. К этому времени результат уже никого не интересовал… Но согласитесь – сколь увлекателен сам процесс!

К 60-м годам прошлого века ЭВМ подверглись усовершенствованию настолько, что уже были способны не только воодушевлять писателей-фантастов и согревать воздух, но и производить некоторые полезные операции, в первую очередь связанные с утомительными математическими расчетами. Вас интересует таблица синусов с точностью до 18 знаков после запятой? Теперь уже не надо напрягать свой мозг – обращаемся за помощью в научно-вычислительную лабораторию, и всего лишь через час распечатка у вас в кармане. И не важно, что вместо синусов вам посчитали косинусы: главное – никаких умственных затрат, да и точность соблюдена!

Повысилась не только производительность процессоров, но и на качественно другой уровень перешли периферийные устройства. Теперь почти пропала необходимость вставки между пользователем и машиной гильдии разношерстных посредников. Человек, имеющий некоторое представление о способах общения с машиной, просто садился за терминал и получал возможность самостоятельно, без какой-либо посторонней помощи вгонять ЭВМ в ступор. Пользователи – люди разные, и далеко не всех интересовали расчеты траекторий баллистических ракет. Огромный пласт людей искал применения для машин в другой не менее важной области – области хранения и обработки больших массивов данных. Судите сами: XX век – век информации, в офисах корпораций, в правительственных учреждениях, в архивах и библиотеках хранились миллионы тонн бумаги с данными о чем угодно, начиная с роста поголовья пингвинов в Антарктике и заканчивая налоговыми декларациями от горячо любимых сограждан. Все это необходимо не просто хранить, а еще и максимально быстро обработать с возможно-

стью получения аналитических выводов, графиков и статистики. До сих пор все эти попытки систематизации колоссальных объемов данных, представленных на бумажных носителях, были в некотором родстве с сизифовым трудом. И вдруг, о чудо, наконец у человечества появился шанс вкатить камень на вершину горы!

Именно с этого момента в сферу обязанностей вычислительных машин, кроме проведения расчетов, вменили еще одну задачу – хранение и обработку больших объемов данных. Тем более что на смену перфолентам и стримерам с магнитной пленкой стали приходить жесткие диски.

> **Определение**
> Данные (data) – информация, представленная в формализованном виде, пригодном для передачи, интерпретации или обработки с участием человека или автоматическими средствами.

Только бесконечно наивный пользователь (метко называемый в народе чайником) может предположить, что ЭВМ сразу же подставила человечеству свое плечо и в мановение ока все бумажные архивы превратились в во всех отношениях совершенные базы данных. Не тут-то было! Вычислительные машины – это лишь инструмент, который работает ровно так, как его научат программисты. Поэтому примерно на стыке 50-х и 60-х годов XX века перед программистами была поставлена задача научить машины обслуживать большие объемы данных.

Электронные картотеки

Мы с вами все учились сами, а некоторые из нас даже пытались учить других. Любой участник образовательного процесса, находящийся в здравом уме и доброй памяти, понимает, что научить можно только тому, что в совершенстве знаешь сам. А теперь ответьте на вопрос: «Что в начале 60-х программисты знали о хранении больших данных?» Прав окажется тот, кто скажет – практически ничего! Так что нет ничего удивительного в том, что на первых этапах становления такой области знаний, как базы данных, все пошли по особому пути, который получил название **систем, основанных на файлах** (file-based system), или просто **системы файлов**. К чему столько скепсиса? Поймете через пару страниц, а пока рассмотрим историю болезни прототипов современных БД – систем файлов.

Особенность человеческого образа мышления заключается в том, что при поиске решений сложных проблем в первую очередь мы пыта-

емся применить уже известные методики. Примеров такого подхода в истории предостаточно. Например, разработчики первых летательных аппаратов весьма настойчиво пытались обучить их махать крыльями – ведь именно так делали птицы. Эффективность подобного решения новаторы обычно проверяли сами, поэтому долго не жили. В большинстве своем программисты – также народ прямолинейный. Создатели прообразов баз данных при поиске решения огляделись вокруг и увидели, что везде, где есть бумажные архивы, данные хранятся в картотеках. Возьмем обычную городскую библиотеку тех лет. Здесь каждой книге соответствует отдельная бумажная карточка, в которой отражены данные об авторе, названии книги, годе издания, месте хранения и т. д. Карточки систематизированы по областям знаний и упорядочены по алфавиту. Любой грамотный посетитель, придя в библиотеку за пару-тройку часов (перелопатив с тысячу карточек), выяснял, что интересующей его книги там нет, и с гордо поднятой головой уходил восвояси… Плохо это или хорошо, но на тот момент времени другого, более рационального решения проблемы архивного дела не существовало. Поэтому воодушевленные программисты, не теряя ни одной секунды на «лишние» размышления, самоотверженно приступили к обучению самолетов махать крыльями – взялись за лобовое проектирование электронных аналогов бумажных картотек.

Замечание

Почему системам, основанным на файлах, мы уделяем столько времени и не переходим сразу к изучению баз данных? По двум причинам. Во-первых, понимание сути недостатков, присущих файловым системам, позволяет избежать их в дальнейшем. Во-вторых, даже сегодня в спектре программного обеспечения есть место для приложений, построенных на основе файлов.

Принцип построения систем файлов

Рассмотрим в общих чертах идею построения систем, основанных на файлах. Допустим, что мы планируем создать программу, хранящую сведения о сотрудниках предприятия. Получивший столь ответственное задание программист поступает следующим образом.

Во-первых, он готовит структуру, подходящую для хранения данных. Для этого программист просматривает список персонала и выясняет максимальное число символов в фамилии, имени и отчестве (допустим, это значения: 20–15–15 байт). Затем программист выделяет ка-

кое-то число байтов для хранения названия должности, даты рождения и остальных подлежащих учету элементов структуры. Узнав все необходимые размерности, разработчик описывает структуру непосредственно в коде программы (рис. 1.1).

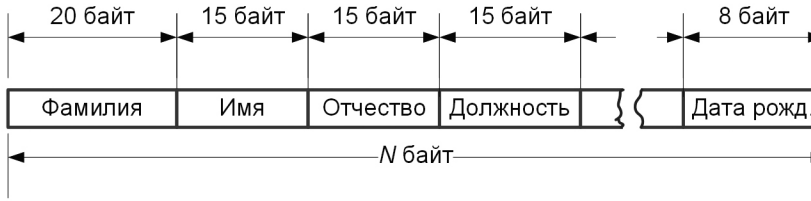

Рис. 1.1. Структура для хранения одной записи в файле

Во-вторых, программистом разрабатываются процедуры, осуществляющие основные операции по работе с файлом. Как минимум это добавление новой записи, редактирование, удаление и просмотр записи. Ни одну из этих операций невозможно осуществить без знания размерности и состава полей исходной структуры. При вставке новой строки в файл к нему следует добавить *N* байт, причем процедура добавления должна знать, что фамилия начинается с 1-го байта, имя – с 21-го и т. д. При просмотре файла необходимо осуществлять последовательные операции чтения порциями, пропорциональными размеру отдельной записи *N* байт; опять же, процедура чтения должна обладать информацией – сколько байт отводится тому или иному полю. В свою очередь, операции редактирования и удаления также не являются исключением из правил и нуждаются в знаниях об исходной структуре. К счастью, эти сведения искать не стоит – все данные о составе полей нашей программе хорошо известны, ведь определение структуры спрятано внутри нее.

Недостатки систем файлов

Поначалу у разработчиков систем, основанных на файлах, дела шли весьма неплохо. Пользователям очень нравилось, что работа с электронными картотеками была схожа с работой с бумажными архивами. В свою очередь, программистам нравилось то, что они сравнительно легко зарабатывают себе на жизнь.

Идиллия продолжалась недолго. Очень скоро, казалось на безоблачном горизонте, забрезжили грозовые тучи – стали проявляться отрицательные стороны лобового подхода разработчиков первых прототипов баз данных. Из всего сонма недостатков особо выделяются пять проблем [25]:

1) зависимость от данных;
2) разделение и изоляция данных;
3) избыточность данных;
4) несовместимость файлов;
5) разрастание количества приложений.

Зависимость от данных. Уже первая проблема, с которой столкнулись разработчики файловых систем, не предвещала ничего хорошего. Допустим, что нам требуется осуществить элементарную операцию – увеличить на один символ размер поля, отвечающего за хранение фамилии. Оказалось, что даже незначительное усовершенствование структуры влечет за собой ком дополнительных трудностей. Судите сами, изменение размера или состава полей структуры вынуждает нас не только переписать исполняемый файл, но и сочинить одноразовую программу-конвертор, которая должна преобразовать старые данные к новому формату.

Разделение и изоляция данных. Системы, основанные на файлах, объединяют в себе десятки отдельных файлов с данными. В одном файле хранятся данные о сотрудниках фирмы, в другом – сведения о клиентах, в третьем – перечень предоставляемых услуг, в четвертом – список заказов и т. д. Для извлечения логически связанных данных (допустим, о клиентах и их заказах) программисту приходилось выдумывать замысловатые алгоритмы синхронного чтения из двух файлов. С увеличением количества файлов, вовлекаемых в итоговый отчет, сложность возрастает в арифметической прогрессии. Задача извлечения взаимоувязанных данных из десятка файлов могла стать непосильной не только для программиста, но и для вычислительных машин тех времен. Масло в огонь подливало еще то, что данные могли быть разделены между отделами и службами предприятия – в отделе кадров находились данные о сотрудниках, в отделе продаж – о заказах и т. п. В 1960-х годах удельный вес предприятий, чьи машины были объединены локальными вычислительными сетями, стремился к нулю, посему данные были еще и изолированы друг от друга. А теперь представьте себе программиста тех времен, мечущегося по организации в надежде объединить разделенные данные в единое целое…

Избыточность (дублирование) данных. С появлением первой микропроцессорной техники многие руководители предприятий стали отказываться от покупок больших ЭВМ и начали отдавать свои предпочтения более дешевым мини-ЭВМ, расставляя их по отделам и службам

своих организаций. Подобное, во многом правильное решение имело и свои отрицательные стороны, одна из них – вынужденный отказ от централизованного хранения данных. Система децентрализованного хранения данных систем, основанных на файлах на нескольких машинах, приводила к тому, что одни и те же сведения повторялась на магнитных носителях многих мини-ЭВМ, разбросанных по учреждению. В этом случае даже был не столь страшен факт избыточности данных, сколь высока вероятность нарушения непротиворечивости данных предприятия – на всех машинах должны храниться идентичные копии данных, но ведь любое изменение данных на одной из ЭВМ никак не отражалось на остальных станциях до тех пор, пока данные не синхронизировали вручную.

В «доисторических» системах файлов избыточность данных вынужденно присутствовала и в рамках одного-единственного проекта. Допустим, что от нас потребуют дополнить программу «Дни рождений сотрудников» еще одним информационным полем – местом работы. В результате в файле появятся многократные дубликаты данных, например Петров – Бухгалтерия, Иванов – Бухгалтерия и т. д. Исследования файловых систем тех времен показали, что до 60 % хранящейся в них данных были избыточны. Учитывая астрономическую стоимость жестких дисков, это весьма непродуктивные расходы.

В свою очередь, избыточность данных порождала целый букет проблем и проблемок. Одна из них – **противоречивость данных**. На одной из рабочих станций предприятия хранится устаревший номер телефона вашего контрагента. На второй этого номера вовсе нет. На третьем компьютере, за счет ошибки оператора, там находится телефон его бабушки. В результате ни один из звонков не достигает цели. Как следствие, фирма терпит убытки.

Аномалии данных. Избыточные и противоречивые данные влекут за собой шлейф дополнительных неприятностей в лице: аномалий добавления новой записи, аномалий редактирования и аномалий удаления. Какая из аномалий способна принести больше печали в наш офис. Судите сами. Допустим, у нашей фирмы появился новый, не жалеющий денег оптовый покупатель. Данные нового клиента следует ввести сразу в несколько систем файлов (отел сбыта, личная картотека главного менеджера, бухгалтерия и т. д.). Если все сделано безошибочно, то все в порядке… но если таких покупателей несколько, то можно гарантировать, что где-нибудь кто-нибудь спутает пару цифр в номерах счетов. В результате платеж уходит на чужой расчетный счет. После долгого

судебного разбирательства и уплаты неустоек вы, наконец, выясняете, в чем причина сбоя, но к этому времени вам уже все равно… С редактированием данных в избыточных системах дела также обстоят далеко не лучшим образом. В идеале можно нанять отдельного сотрудника, задачей которого станет регулярная пробежка по всем структурным подразделениям компании с целью исправить почтовый адрес (номер телефона, дату рождения, номер счета или что-нибудь в этом духе) главного спонсора фирмы. В результате поздравительная открытка, отправленная в канун очередного юбилея, не попадет к адресату, а в отместку «благодарный» юбиляр не перечислит вашей компании давно обещанные (и так необходимые сейчас) финансовые влияния. Впрочем, считайте, что вам крупно повезло, ведь уязвленное самолюбие может привести и к более серьезным последствиям… Аномалия удаления в состоянии принести не меньшие неприятности. Как вы думаете, как скажется на финансовом состоянии фирмы тот факт, что она станет выплачивать ежегодные премии давно уволенному менеджеру? Это печальное событие произойдет только потому, что в бухгалтерии забудут вычеркнуть всего одну строку с данными.

Несовместимость файлов. Структура файлов с данными определялась не только разработчиками программного обеспечения, но и языками программирования, состоящими на вооружении в тех или иных организациях. Построение файла, описанного на языке Algol, могло принципиально отличаться от структур, генерируемых средствами PL/1, ADA или какого-нибудь еще средства разработки приложений тех времен. Более того, дополнительные ограничения вносились из-за особенностей архитектурных решений, принятых в основу построения тех или иных ЭВМ. Помножьте это на специфичные черты различных операционных систем. Как следствие выходящие из-под «пера» программистов файлы зачастую становились несовместимыми, хотя и содержали практически идентичное описание данных.

Разрастание количества приложений. Сами по себе данные не представляют никакого интереса. Представьте, что у вас имеется файл с несортированными телефонными номерами жителей миллионного города – это хорошая новость. А теперь плохая – у вас нет средств упорядочивания и поиска данных. В результате цена таким данным – ломаный грош, ну-ка найдите номер телефона гражданина Иванова, затерявшегося где-то среди сотен тысяч других номеров… Данные надо не только хранить, но и уметь представлять пользователю в удобном формате. А пожеланий у пользователей не счесть, одним требуется,

чтобы списки заказов упорядочивались по алфавиту, другим – по дате заказа, третьим хотелось бы, чтобы имелась возможность сортировки записей по денежной сумме. Старуха, затребовавшая от Золотой рыбки перечень услуг (начиная от тривиального корыта и заканчивая царским троном), – просто ангел в сравнении с запросами к данным у биржевого аналитика, главного бухгалтера завода или заведующего гипермаркетом. Идеи и пожелания пользователей сыплются на программиста как из рога изобилия, и никто, кроме него, не ведает, что каждый новый запрос приводит к цепной реакции – бесконечной переработке исходного приложения. В конце концов, головная программа обрастает скопищем утилит и «утилиток», что рано или поздно (а главное – необратимо) приведет проект к коллапсу.

Пути устранения недостатков систем файлов

Сегодня системы, основанные на файлах, практически не используются, исключение составляют состоящие из одного-двух файлов данных небольшие по числу записей хранилища. Дабы не повторить ошибки программистов тех времен, проектировщики БД сделали нужные выводы.

Во-первых, разработчики в принципе отказались от хранения физической структуры данных в коде приложений. Вместо этого описание данных стало выноситься в отдельное хранилище, называемое **системным каталогом** (system catalog). Таким образом, во всех современных БД, помимо собственно хранимых в них данных, еще имеются метаданные (данные о данных). Если внешняя программа обладает возможностью чтения метаданных, то она без труда сможет получить доступ к хранимой в БД информации.

Во-вторых, стали предприниматься активные попытки стандартизировать способы описания и хранения данных. Наличие единого для всех разработчиков стандарта значительно упростило доступ к данным.

В-третьих, возникла необходимость создания единого универсального языка, позволяющего производить с данными наиболее важные операции: вставки, редактирования, удаления и просмотра.

В результате на смену морально устаревшим системам файлов пришли свободные от недостатков своих предшественников базы данных.

Как видите, в составе баз данных основную роль играют структуры и описывающие эти структуры метаданные, кроме того, в БД задействуются индексы, представления, хранимые процедуры, триггеры и т. п. (на рисунке мы их обозначили многоточием). А роль приложения сведена

к минимуму, т. к. основная логика управления данными осуществляется силами самой БД. В свою очередь, в системах файлов приложения важны не менее самих данных, ведь, кроме них, никто не в состоянии предоставить доступ к данным.

Что такое база данных?

На протяжении всей главы мы смело оперировали таким важным понятием, как «база данных», но пока наполняли его интуитивным содержанием. Теперь настал тот час, когда мы введем определение БД. Сначала обратимся к стандарту.

В соответствии с ГОСТ 34.321–96, устанавливающим эталонную модель управления данными, под базой данных понимается «совокупность взаимосвязанных данных, организованных в соответствии со схемой базы данных таким образом, чтобы с ними мог работать пользователь» [6].

Несмотря на то что приведенное в ГОСТ определение следует считать официальным, на страницах этой книги никто не запрещает нам подойти к определению БД творчески и немного его доработать, при этом не искажая изначального смысла, заложенного в ГОСТ (тем более что со времен утверждения стандарта прошло немало времени).

> **Определение**
> База данных (database) – это организованная совокупность совместно используемых логически связанных данных и описаний этих данных, относящаяся к определенной предметной области, предназначенная для удовлетворения информационных потребностей организации.

Дополним нашу версию определения БД рядом комментариев.

1. База данных представляет собой долговременное хранилище данных, структура которого определяется на этапе проектирования БД.
2. Находящиеся в БД данные являются общекорпоративным ресурсом, в котором вместо отдельных файлов, разбросанных по отделам и службам предприятия, данные собраны вместе с минимальной избыточностью.
3. В БД хранятся не только рабочие данные, описывающие какую-то предметную область, но и метаданные, предназначенные для

описания структуры самих данных. Такую важную способность БД часто называют самодокументированием.

4. Потребителем данных может выступать не только человек, но и программно-аппаратные комплексы (станки с числовым программным управлением, автоматизированные системы управления, робототехнические системы и т. д.), именно поэтому в определении говорится об удовлетворении информационных потребностей не просто пользователей, а организации в целом.

Эволюция моделей БД

Идея построения первых БД на принципах электронных картотек просуществовала сравнительно недолго и была отвергнута. Ограничения систем, основанных на файлах, вошли в непримиримое противоречие с возрастающими требованиям к хранению и обработке больших массивов данных, и, как следствие, проигравший должен был уйти с арены. Однако что можно было бы предложить взамен?

В 1960-х годах разработчикам программного обеспечения пришлось взять тайм-аут и задуматься над путями устранения возникших трудностей. Хорошим стимулом тому были грандиозные проекты второй половины века. Чего только стоила амбициозная цель НАСА первыми отправить астронавта на Луну. США, получившие поучительный щелчок по носу от СССР на первом этапе космической гонки, не жалели ни денег, не ресурсов для взятия реванша. А борьба за космос – это не только битва реактивных двигателей. Здесь в равной степени важны достижения практически во всех областях науки и техники, и среди них далеко не на последнем месте стоят компьютерные технологии. Попробуйте представить себе астронавта с логарифмической линейкой, прикидывающего в уме, сколько надо подать топлива в форсунки двигателя, дабы в очередной раз не проскочить мимо спутника нашей планеты… Поэтому американскому космическому агентству потребовались производительные малогабаритные ЭВМ и современное программное обеспечение. В эту отрасль начали вкачиваться огромные денежные суммы. В результате над проектированием первых систем баз данных был сосредоточен внушительный спектр корпораций и научно-исследовательских лабораторий. Так что, если хотите, движущей силой современных баз данных в 1960-х годах стали уязвленное самолюбие, помноженное на отменное финансирование.

Спустя высадки человека на Луну минуло полвека, неплохой срок для подведения итогов развития баз данных. Возьмем на себя смелость утверждать, что за этот период базы данных шагнули вперед так же далеко, как и космонавтика. На рис. 1.2 отражены этапы развития моделей баз данных. Это так называемые модели реализации, т. е. модели, ориентированные на получение ответа на вопрос: «Каким образом следует описывать структуры данных?» Единственное исключение составляет понятийная модель «сущность–связь», это ближайший союзник реляционной модели, но отвечающий не за реализацию, а за логику будущей БД.

Рис. 1.2. Эволюция моделей реализации данных

Системы, основанные на файлах, лишь с большой натяжкой можно причислять к базам данных, поэтому мы отнесем их к предварительному этапу (если хотите – к этапу наивного проектирования). Появление первых моделей БД (сетевая и иерархическая модели) заместило файловые системы, этот условный «дореляционный» период продолжался с середины 60-х по первую половину 70-х годов XX века. Началом его конца стала публикация знаменитой статьи Э. Кодда о реляционных банках данных. Закат одного дня – это лишь прелюдия нового рассвета. Эпоха реляционной модели настала вместе с выходом в свет знаменитого прототипа реляционной базы данных – System-R.

Замечание

К моделям «дореляционного» периода относится еще один вид БД – системы с инвертированными списками. Указанная модель реализации БД не получила широкого распространения, поэтому в этой книге мы ее не рассматриваем.

Необходимость моделирования

Говоря об эволюции баз данных, мы уже несколько раз употребили термин «модель». Почему так акцентируется на этом внимание? Для чего нужен процесс моделирования при создании БД?

Построение различного рода моделей широко применяется в различных областях науки и техники. В авиастроении, прежде чем поднять в небо дорогостоящий самолет, его уменьшенная модель проходит исследование в аэродинамической трубе. Конструкторы летательного аппарата выясняют, насколько удачны контуры будущего истребителя или пассажирского лайнера, не рискуя жизнями экипажа. Физики и астрономы, изучая особенности протекания ядерных реакций на Солнце, не имеют даже теоретической возможности потрогать светило руками. Поэтому они моделируют термоядерную реакцию в микроскопических масштабах в своих лабораториях. Экономисты ведущих стран строят математические модели поведения финансовой системы страны, на которой апробируются их законодательные инициативы, – это гораздо безопаснее, чем опыты на своих согражданах. Несмотря на разную направленность, у всех рассмотренных выше случаев есть объединяющая черта – модель создается в том случае, когда проведение исследований на реальном объекте невозможно (по финансовым, экономическим, этическим, техническим или любым другим причинам).

Вне зависимости от моделируемого объекта создателям модели необходимо выполнить очень важное условие – модель должна быть адекватна реальности, иначе результаты исследования можно выбросить в мусорную корзину. Вполне естественно, что при подготовке модели самолета для продува в аэродинамической трубе не стоит тратить время на создание миниатюрных двигателей, прокладку электропроводки и размещение кресел внутри салона. Это не столь важно в сравнении с безукоризненным соблюдением пропорций фюзеляжа, крыльев и хвостового оперения летательного аппарата. Поэтому в процессе моделирования ученые отбрасывают малозначимые детали, уделяя первоочередное внимание ключевым характеристикам модели, в последнем примере с точки зрения аэродинамики.

На страницах этой книги мы еще много раз будем говорить о моделях баз данных и, в частности, о доминирующей сегодня реляционной модели. Теперь мы знаем, что модель базы данных призвана отражать реальные и виртуальные объекты окружающего мира, информацию о которых мы планируем хранить и обрабатывать в разрабатываемых информационных системах. Чем точнее модель отражает действительность – тем она полезнее и, как следствие, лучше база данных.

На сегодняшний момент времени подавляющее большинство современных баз данных ориентировано на реляционную модель. Это простейшие однопользовательские системы типа Microsoft Access и SQLite, и сложные клиент-серверные системы SQL Server компании Microsoft, Informix, Oracle, InterBase, MySQL. Список можно продолжить… Но так было не всегда. До второй половины семидесятых годов прошлого века рынок баз данных был поделен примерно поровну между программными продуктами, исповедующими иерархическую или сетевую модель представления данных.

Иерархическая модель

Своим появлением на свет иерархическая модель данных обязана лунному проекту Apollo и двум американским компаниям: Rockwell International и IBM. Президент США Дж. Ф. Кеннеди пообещал высадить американского астронавта на Луну к концу 1960-х. .Для доставки человека на поверхность спутника нашей планеты требовалось не только рассчитать траекторию полета, но и обработать огромное количество вспомогательных данных, начиная от расхода кислорода астронавтами и заканчивая учетом заказов на уборку помещений НАСА. Итак, в начале 1960-х North American Aviation (именно так в те годы именовалась Rockwell) стала генеральным подрядчиком по разработке программного обеспечения для обслуживания всего проекта. В результате пятилетнего труда специалисты компании предложили заказчику программу с мудреным названием «Обобщенный метод доступа и модификации» (Generalized Update Access Method, GUAM), способную хранить данные в виде перевернутой иерархической структуры (рис. 1.3). В это же время к Rockwell присоединился субподрядчик IBM, преследовавший еще одну заветную цель – разработать первый коммерческий проект баз данных. Совместное детище двух компаний появилось на свет во второй половине 1960-х и получило название «Информационно-управляющая система» (Information Management System, IMS). В отличие от всех своих конкурентов, проект IMS

приобрел одно уникальное по тем временам преимущество – система позволяла работать с данными не только на физическом, но и на логическом уровне. Новый программный продукт очень скоро стал столь популярен, что без него не обходился ни один крупный мейнфрейм вплоть до конца 70-х годов прошлого века.

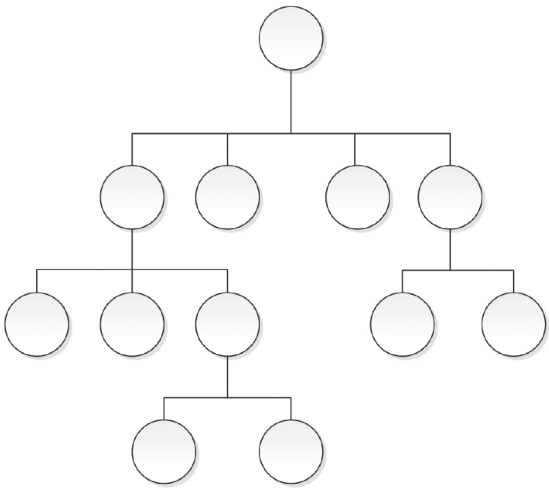

Рис. 1.3. Иерархическая модель данных

Суть иерархического хранения данных заключается в применении инвертированной древообразной структуры. В наивысшей точке располагался корень дерева, ниже – дочерние узлы (листья дерева). Все узлы связывались друг с другом благодаря сложной системе указателей. Каждый из узлов, в свою очередь, мог являться родительским по отношению к одному или нескольким нижерасположенным узлам, но у дочернего узла не может быть более одного родителя. Таким образом, в иерархической модели реализована одна из наиболее часто встречающихся в реальном мире связей между сущностями – связь «один ко многим». Например, в городе находится много компаний, в компании образовано много отделов, в отделе работает много сотрудников. Подобное решение существенно снижает остроту проблемы избыточности данных. Проект базы данных «отдел – сотрудник», реализованный средствами иерархической модели, мог бы выглядеть так, как представлено на рис. 1.4.

Это несколько усовершенствованное решение предусматривает описание дополнительных данных – места работы сотрудника. Заметьте, что узлы сотрудников выступают в качестве подчиненных по отноше-

нию к узлам отделов и не могут существовать без них. При желании схему можно и развить, например подчинив сотрудникам узлы с заказами, описанием работ, номерами счетов и т. п. Подобное развитие не требует кардинального изменения уже существующих узлов дерева, мы лишь добавляем очередные узлы на новом нижележащем уровне.

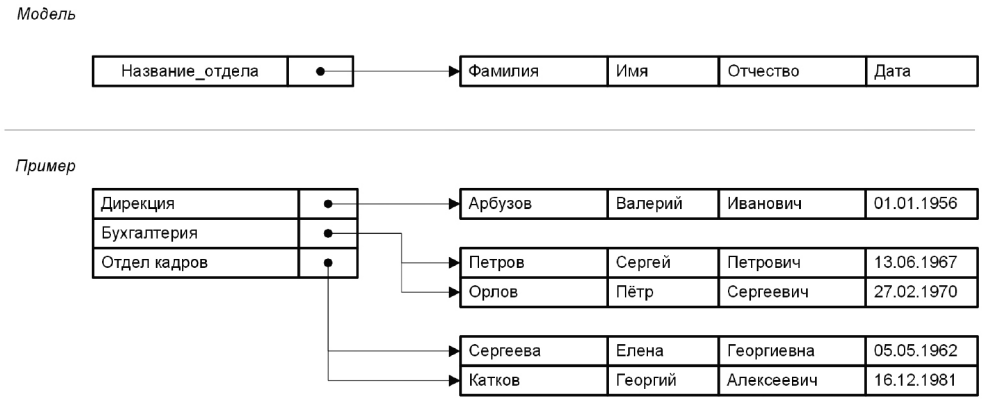

Рис. 1.4. Пример работы иерархической БД

По сравнению с системами файлов иерархическая модель весьма неплоха, вот только ключевые ее достоинства:

1) простота понимания структуры данных. Иератическое построение данных интуитивно понятно, что существенно упрощает проектирование БД;
2) целостность данных. Иерархические БД представляли собой неразрозненные приложения и файлы. Все данные находились под контролем одной системы управления базами данных, которая не допускала некорректные действия с записями (например, удаление родительского узла, у которого оставались «осиротившими» дочерние элементы);
3) независимость данных. Данные больше не принадлежат одному приложению. Наличие системного каталога позволяет работать с БД приложениям, умеющим читать метаданные;
4) безопасность данных. Контролируемый доступ к данным осуществляется с помощью СУБД, в которую закладывается предпочтительная политика безопасности.

Однако, поборов недостатки систем, основанных на файлах, иерархическая модель приобрела новые. Вот только некоторые из них:

1) ограничения в организации отношений между сущностями. Иерархическая модель позволяет организовать последовательную связь «один ко многим» между данными, но не в состоянии реализовать отношения «многие ко многим»;
2) структурная зависимость. Иерархическая структура предполагала, что физически данные также станут храниться в виде дерева. Серьезное изменение структуры (например, переподчинение узлов) могло привести к тому, что прикладные приложения теряли возможность навигации по данным;
3) сложность разработки прикладного программного обеспечения (ППО). Разработчик ППО должен знать особенности физического хранения данных, иначе он мог просто заблудиться в запутанной системе указателей.

Ко всему прочему иерархическая модель не была стандартизирована. Как следствие всегда существовала проблема переносимости данных между приложениями различных разработчиков.

Замечание

Иерархическая организация данных очень удобна при организации поиска данных, запрашиваемые сведения уже сгруппированы по соответствующим ветвям дерева, и нам остается лишь спускаться от корня схемы к требуемому листу.

Сетевая модель

Практически параллельно с North American Aviation и IBM свои собственные разработки вела еще одна серьезная американская корпорация – General Electric. Ею в 1964 был выпущен релиз первой сетевой базы данных IDS (Integrated Data Store). Так же, как и разработчики иерархической модели, General Electric работала в условиях отсутствия утвержденного стандарта на описание среды БД.

Заметим, что когда мы говорим о «сетевой» модели, в виду имеется особенность организации данных, а не способ объединения компьютеров в сеть. В данном случае за основу была взята не древовидная модель данных, а структура, построенная на основе графа (рис. 1.5).

По сравнению с иерархической, сетевая модель имеет одно серьезное преимущество: она позволяла назначать произвольное количество связей между узлами графа. Поэтому она в состоянии создавать базы данных, более точно отражающие связи реального мира, в частности в сетевых БД без особого труда можно было формировать отношения

«многие ко многим» или замыкать связь узла на себя самого. При удалении записи в сетевой БД уничтожается только этот соответствующий ей элемент и связь с ним, все остальное остается на месте. Это разительное отличие от иерархически организованной БД, ведь здесь при удалении одного узла удалению подлежала вся нижерасположенная ветвь дочерних элементов или требовалось провести относительно сложную процедуру переподчинения дочерних узлов. Впрочем, простота перестроения структуры графа имеет и свои подводные камни, ведь любое непродуманное изменение связей может привести к нарушению целостности данных.

Замечание

Ключевое отличие между иерархической и сетевой моделями данных заключается в том, что в древообразных структурах дочерний узел может обладать только одним родительским узлом, а в сетевой модели ограничение на количество родительских элементов отсутствует.

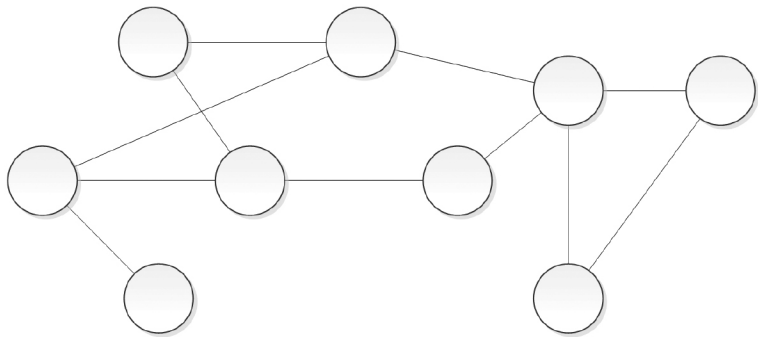

Рис. 1.5. Сетевая модель данных

К сожалению, сетевая модель также не свободна от недостатков:

- большое количество произвольных связей повышает сложность схемы БД и как следствие вызывает дополнительные трудности при обеспечении целостности данных;
- сложность разработки прикладного программного обеспечения.

Замечание

Говоря о сетевой модели данных, обязательно следует упомянуть заслуги руководителя проекта IDS Чарльза Вильяма Бэчмэна (Charles William Bachman), за свой вклад в науку в 1973 году он был удостоен премии Тьюринга.

Говоря о сравнительных характеристиках рассмотренных иерархической и сетевой моделей, сразу отметим, что они свободны от недостатков систем, основанных на файлах. Все это достигнуто благодаря тому, что иерархическая и сетевая модели предусматривают существование некоторого внешнего хранилища (системного каталога), в котором размещаются описания данных. Современные системы управления базами данных размещают в нем всю информацию, необходимую для нормальной жизнедеятельности базы данных, включая описание используемых типов данных, ограничений на вводимые данные, отношений, связей между отношениями и многое-многое другое. Благодаря наличию системных метаданных достигается архиважный для базы данных фактор физической независимости приложений от данных. Теперь грамотно спроектированное приложение после ознакомления с системным каталогом способно получать доступ практически к любому набору данных.

Попытки разработки стандарта БД

Во многом недостатки первых БД были предопределены вакуумом в области стандартов. Разработчики программного обеспечения фактически работали в голом поле, полностью полагаясь на свою интуицию. Поэтому уделим несколько минут внимания знакомству со сложившейся к середине 1960-х годов обстановкой в области стандартизации БД.

А ситуация напоминала извечный философский спор о первичности курицы или яйца. Корректный стандарт, определяющий способы хранения и описания данных в будущих БД, не мог появиться на пустом месте, для этого был необходим опыт реализации и эксплуатации подобных баз. Одновременно с этим любая попытка построения СУБД в условиях отсутствия стандартов была обречена на провал. Существовал ли выход из такого положения? Оказывается, да.

Острое желание победить в лунной гонке, помноженное на завидные финансовые возможности, позволило США пойти по пути одновременного создания курицы и яйца. Компании North American Aviation, IBM, General Electric и десятки их помощников уже приступили к проектированию пилотных версий БД. Одновременно с этим в 1965 году при непосредственном покровительстве правительства США на конференции CODASYL (Conference on Data Systems Languages) была сформирована рабочая группа List Processing Task Force, переименованная в 1967 году в группу Data Base Task Group (DBTG). Всю свою энергию DBTG посвятила вопросам определения спецификаций среды, которая допускала бы

разработку баз данных и управление данными. Группа сумела написать два научных отчета, промежуточный в 1969 и итоговый в 1971 году. Заметьте, что к этому времени астронавты уже оставили свои следы на луне.

Несмотря на упорные старания группы DBTG, ее труд не смог стать стандартом, отчеты DBTG не удовлетворили Национальный институт стандартизации США (American National Standard Institute, ANSI). Вместе с тем работа ни в коем случае не прошла даром и подготовила фундамент для будущих исследований. Сложно в двух предложениях резюмировать более чем пятилетнюю работу специалистов, но отметим следующее. Группа DBTG признала необходимость использования двухуровневого подхода, построенного на основе использования системного представления и пользовательских представлений. Если уж совсем просто – БД должны создаваться так, чтобы пользователю даже не имело смысла интересоваться, каким образом данные хранятся физически, ему надо лишь обладать возможностью ими пользоваться.

В 1975 году Комитет планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США предложил свое видение модели БД – 3-уровневую архитектуру ANSI/X3/SPARC. Ближе всего к данным предлагалось ввести первый уровень абстракции – внутренний (рис. 1.6). На внутренний уровень возлагались задачи управления физическим хранением данных в памяти ЭВМ. Над внутренним уровнем расположился очередной уровень абстракции – концептуальный. Этот уровень уже не нес ответственности за хранение данных на носителях информации. Комитет SPARC предлагал возложить на концептуальный уровень обязанности за описание логической структуры всех данных, в частности логический смысл хранимых в БД данных, связи между данными, ограничения на данные, правила поддержки целостности и т. п. Наконец, третий по счету слой архитектуры ANSI/X3/SPARC получил название внешнего уровня. Внешний уровень отвечал за организацию интерфейса между человеком и БД. Конечный пользователь нашего программного продукта может быть высококлассным программистом, а может и оказаться «чайником». Уровень должен уметь предоставить и тому, и другому соответствующие его потребностям и умениям инструменты по доступу и управлению данными.

Таким образом, архитектура ANSI/X3/SPARC развивает задумку группы DBTG. И там, и здесь во главу угла ставится обеспечение независимости от данных. Независимость означает, что изменения на нижних уровнях никак не влияют на верхние уровни. Только в последнем случае

комитет SPARC пошел дальше и предложил три слоя абстракции. Благодаря этому модель обеспечивала не только физическую, но и логическую независимость от данных.

Впрочем, дополнительный уровень не решил проблему, и модель ANSI/X3/SPARC все равно не сумела стать стандартом. Несмотря на то что труды группы DBTG и комитета SPARC не привели разработчиков к запланированному результату и не стали стандартом, предложенные ими модели оказали значительное влияние на специалистов-практиков, в особенности на создателей первой реляционной БД, получившей название System-R.

Рис. 1.6. Трехуровневая архитектура ANSI/X3/SPARC

Реляционная модель

Довольно скоро на смену иерархической и сетевой моделям пришла принципиально новая модель данных – реляционная. Реляционная модель в следующих главах будет рассмотрена весьма подробно, а пока упомянем лишь то, что она основана на принципе выявления подлежащих описанию в БД сущностей и связей между ними. По сравнению со своими товарками она отличается высокой гибкостью, здесь программист гораздо в меньшей степени ломает голову над механизмом управления данными. Эта задача автономно решается ядром СУБД. Реляционные БД снабжены специализированным языком SQL, который достаточно легок в освоении. Таблицы реляционной модели жестко структурированы, что упрощает их обслуживание.

Теория реляционной модели в первую очередь опиралась на вышедшую в 1970 году статью[1] Э. Кодда (E. F. Codd), а первый существенный прикладной результат пришел в 1976-м. В этом году в исследовательской лаборатории корпорации IBM, расположенной в городе Сан-Хосе штата Калифорния, на свет появился прототип современных реляционных СУБД – проект System-R. Руководителем проекта был Мортон Астрахан (Morton M. Astrahan).

Проект System-R преследовал цель доказать практичность реляционной модели, что достигалось посредством реализации предусмотренных ею структур данных и требуемых функциональных возможностей. На основе этого проекта был разработан структурированный язык запросов (в ту пору названный SEQUEL), который несколько позднее (в 1986 г.) стал стандартом SQL.

На базе System-R впоследствии (в 1975–1979 гг.) был создан первый успешный коммерческий реляционный продукт фирмы IBM – DB2. Говоря о DB2, нельзя не упомянуть одного из ее авторов – Криса Дж. Дейта (Chris J. Date). На сегодняшний день это ведущий специалист по реляционной модели данных, по всему миру широко известна и многократно переиздавалась его книга «Введение в системы баз данных» [19], на которой выросло не одно поколение программистов баз данных.

Практически параллельно с System-R над схожим проектом в Калифорнийском университете работали Майкл Стоунбрейкер (Michael Stonebraker) и Юджин Вонг (Eugene Wong). Проект получил название INGRES (INteractive GRaphics REtrieval System) и был доведен до реально работающей одноименной СУБД. Позднее на фундаменте INGRES была разработана СУБД Postgres (от лат. *post* и *Ingres*), которая, в свою очередь, стала родоначальником очень популярной сегодня СУБД PostgreSQL.

Как и все в нашем мире, реляционная модель данных далеко не идеальна. По существу, это компромиссное решение между потребностью отражать сущности реального мира и связи между ними и ограниченными возможностями математической теории множеств, переложенной на программный код. Любой компромисс предполагает появление множества малоприятных ограничений. Так, из-за борьбы с избыточностью данных в реляционной БД (процесс нормализации) сведения «размазываются» по нескольким отношениям (таблицам). Как следствие для получения сводного отношения приходится собирать данные по крохам и осуществлять множество соединений. Чем больше соединений следует провести, тем больше временных и ресурсных затрат.

[1] *Codd E. F.* A Relational Model of Data for Large Shared Databanks. Communications of the ACM, 06.1970. P. 377–387.

Другая проблема реляционной модели заключается в особенностях организации связи между отношениями. Для моделирования всего многообразия взаимодействия между сущностями реального мира в нашем распоряжении имеется лишь одна конструкция «один ко многим». Для создания отношения «многие ко многим» приходится выкручиваться за счет введения дополнительной соединительной таблицы и применения двух типов связей «один ко многим». Но на этом проблема построения связей не исчерпывается. Реляционная модель попросту бессильна, когда следует отразить смысловую нагрузку связи. Что делать, когда семантика связи между сущностями различается? Мы с вами видим разницу между глаголами «обладает», «подчиняется» или «управляет», а реляционная модель нет…

Замечание

В отличие от однотипных записей реляционных БД, записи в иерархической и сетевой моделях могут обладать различной структурой, т. е. содержать различное число разнотипных полей.

Спустя пять лет после того, как мир впервые услышал о реляционной модели, ученый Питер Пин Шань Чен дополнил теорию моделирования данных доктора Кодда весьма удобным описанием модели, получившей название ER-модель. Об этой модели мы поговорим в 6-й главе книги, а пока отметим то, что Питер Чен предложил простой и эффективный способ представления сущностей, атрибутов и связей между ними в виде наглядных диаграмм. В отличие от сетевой, иерархической и реляционной моделей (относящихся к классу моделей реализации), модель Чена принадлежит к разряду концептуальных (понятийных). Другими словами, ER-модель отражает логическую природу представления данных.

Объектно-ориентированная модель

В середине 1980-х годов на рынке программных продуктов стали активно появляться программные средства, построенные на основе объектно-ориентированной парадигмы. В ту пору все разработчики как заклинание повторяли новые термины: абстрагирование, инкапсуляция, модульность, иерархичность. Теперь на верхушку пирамиды вознесен Его Величество Объект. Объект в технологии объектно-ориентированного программирования выступает удобной абстракцией объектов из реального мира. Он позволяет описывать все, начиная от

авторучки и заканчивая тепловозом. Главное, чтобы программист грамотно сконструировал исходный класс. Программный объект обладает свойствами, методами и способен реагировать на события. Одним словом, достоинств столько, что многие начали говорить о том, что настал час задуматься о замене реляционной модели данных на перспективную объектно-ориентированную.

Одним из первых теоретических изысканий считается семантическая модель данных (Semantic database model, SDM), предложенная М. Хаммером (M. Hammer) и Д. Маклеодом (D. McLeod) и опубликованная в статье в 1981 году.

Желание создать что-то новое было столь острым, что в первой половине 90-х годов группой под названием Object Database Management Group были предложены рекомендации по созданию объектно-ориентированных баз данных ODMG-93. Примерно тогда начали появляться мнения о закате реляционной модели данных. Однако до сегодняшнего дня реализовать рекомендации ODMG-93 в полном объеме не удалось. Основная причина этого – отсутствие развитого математического аппарата, способного описывать данные в формате, приемлемом для объектно-ориентированной теории.

С точки зрения точности моделирования реального мира объектно-ориентированная модель данных по всем статьям сможет превзойти реляционную, ведь в идеале она должна быть способной адекватно выразить информационные структуры любой сложности. То же самое можно сказать и о реализации связей между объектами – вариантов не счесть. Самым главным достоинством СУБД очередного поколения может стать тот факт, что проведение логических операций над данными многократно усилится за счет возможностей ООП. Но, к сожалению, даже спустя три десятилетия мы не увидели коммерчески успешного программного продукта с этикеткой «объектно-ориентированная СУБД». Так что пока можно говорить только о взаимной интеграции реляционной модели и технологии ООП.

Замечание

Наибольшие противоречия между специалистами по базам данных вызывает объектно-ориентированная модель баз данных (ООМБД). Ряд авторов утверждает, что ООМБД (object-oriented database model, OODM) приходит на смену реляционной и представляет собой СУБД 3-го поколения, а другие, например Крис Дж. Дейт, вместо термина «объектно-ориентированная модель» используют термин «объектно-реляционная система» [19], тем самым подчеркивая, что объектно-ориентированные

системы лишь дополнили реляционную модель. Так что слухи о кончине реляционной модели оказались сильно преувеличены. И на сегодняшний день реляционная модель является доминирующим направлением развития всех современных систем баз данных. Практическим подтверждением слов К. Дейта является реляционная СУБД Oracle 11.0, поддерживающая элементы объектно-ориентированного подхода.

Существенный вклад в разработку объектно-реляционной системы внес американец Майкл Стоунбрэкер (Michael Stonebraker). Майкл в 1970-е работал над проектом Ingres, в 80-е возглавлял проект Postgre, в 90-е ряд его идей был реализован в СУБД Informix Universal Server. Стоит отметить, что Стоунбрэкер (так же, как и Крис Дж. Дейт) с крайней осторожностью применял термин «объектно-ориентированная модель», предпочитая вместо него говорить о системах баз данных будущего поколения.

Слабоструктурированные данные

В тех ситуациях, когда на первое место выступает не столько решение задачи хранения записей, сколько простота организации и универсальность обмена данными, стоит обратить внимание на идею работы со слабоструктурированными данными. В качестве естественного способа хранения подобных данных выступает обычный текстовый формат. Что можно придумать проще?

Для обслуживания слабоструктурированных данных в 1998 году разработан отдельный стандарт – расширяемый язык разметки (eXtensible Markup Language, XML). Особо подчеркнем, что слабоструктурированные данные в формате XML не нуждаются в услугах отдельных СУБД, а для обработки XML разработан специальный прикладной интерфейс программирования.

Документ-ориентированная модель

Документ-ориентированная (document-oriented) модель данных – явление сравнительно новое, появившееся в начале 2000-х. Данная модель выступает едва ли не антиподом реляционной теории и опирается на идею хранения иерархических структур данных, в которой каждый элемент представляет собой целостный, а не «размазанный» по двумерным отношениям, как в реляционной модели, документ. Документы хранятся не в таблицах, а в специальных хранилищах документов (document store). Основным инструментом по доступу и управлению данными выступает декларативный язык NoSQL (Not only SQL).

Документ-ориентированные СУБД свободны от ключевого недостатка их реляционных коллег – существенных затрат на сборку данных из нескольких таблиц в единое целое. В этом просто нет необходимости, ведь документ никто и не собирался разделять. Обратной стороной медали выступает ограниченность синтаксических конструкций по управлению данными языка NoSQL.

Классическими примерами документ-ориентированных СУБД можно считать MongoDB, CouchDB, Couchbase, MarkLogic.

> **Замечание**
> Рассмотренные в главе модели БД являются наиболее востребованными, но далеко не единственными. Помимо них, можно упомянуть хранилища «ключ–значение», графовую и столбцовую модели.

Резюме

За сравнительно недолгий период существования модели реализации баз данных совершили несколько эволюционных шагов – от первых систем, основанных на файлах современных моделей. Отказ от систем файлов и переход к базам данных позволили получить ряд существенных преимуществ: непротиворечивость данных, целостность данных, совместное использование данных, контролируемая избыточность и высокая безопасность данных.

Совместно с моделями реализации развивалась и теория баз данных, появлялись на свет принципиально новые языки программирования (например, ставший сегодня стандартом структурированный язык запросов SQL, языки NoSQL и NewSQL) и особый вид программного обеспечения – системы управления базами данных (СУБД).

Вопросы для самопроверки

1. Перечислите ограничения, присущие системам, основанным на файлах.
2. Что стало причиной большинства ограничений систем, основанных на файлах?
3. Для чего предназначен системный каталог?
4. Что такое метаданные?
5. Что понимается под термином база данных?
6. Поясните, почему база данных является моделью.
7. Прокомментируйте этапы развития моделей данных.

8. Почему в вопросе хранения данных так важна стандартизация?
9. Каковы состав и назначение уровней архитектуры ANSI/SPARC?
10. Перечислите достоинства и недостатки иерархической модели данных.
11. Перечислите достоинства и недостатки сетевой модели данных.
12. Для чего нужны слабоструктурированные данные?
13. Что поставлено в основу объектно-ориентированной модели данных?
14. В чем заключается ключевое отличие в подходе хранения данных между реляционной и документ-ориентированной моделью?