Лекция 0. Введение в БД. Управление данными: Прошлое, Настоящее и Будущее

Взгляд в историю: шесть поколений систем управления данными, начиная от ручных методов, через несколько стадий автоматизации управления данными.

- 0 поколение: ручные менеджеры записей (4000 г. до н. э. – 1900)
- 1 поколение: механизированные менеджеры записей (1900-1955)
- 2 поколение: программируемое оборудование обработки записей (1955-1970)
- 3 поколение: оперативные сетевые базы данных (1965-1980)
- 4 поколение: реляционные базы и архитектура клиент-сервер (1980-1995)
- 5 поколение: мультимедийные базы данных (1995-...)

В управлении данными имелось шесть разных фаз.

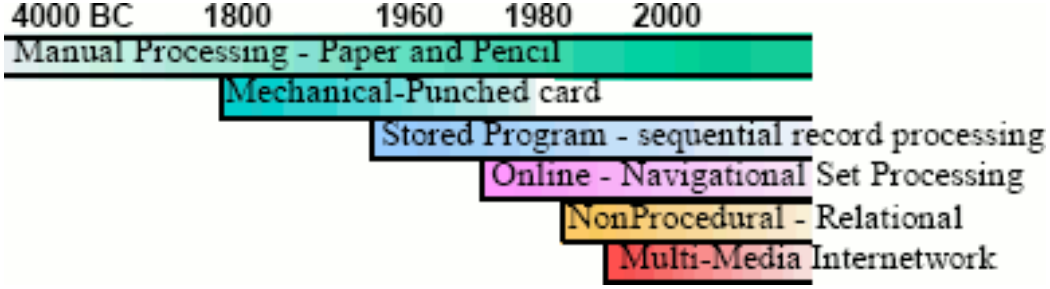


Рис. 1. Шесть поколений управления данными.

Вначале данные обрабатывались вручную. На следующем шаге использовались оборудование с перфокартами и электромеханические машины для сортировки и табулирования миллионов записей. На третьей фазе данные хранились на магнитных лентах, и сохраняемые программы выполняли пакетную обработку последовательных файлов. На четвертой фазе было введено понятия схемы базы данных и оперативного навигационного доступа к данным. На пятой фазе был обеспечен автоматический доступ к реляционным базам данным и была внедрена распределенная и клиент-серверная обработка. Теперь мы находимся в начале шестого поколения систем, которые хранят более богатые типы данных, в особенности, документы, изображения, аудио- и видеоданные. Эти системы шестого поколения представляют собой базовые средства хранения для появляющихся приложений Internet и intranet.

0.0 Нулевое поколение: ручные менеджеры записей (4000 г. до н. э. – 1900)

В первых известных письменных свидетельствах описывается учет царской казны и налогов в Шумере. Поддержка записей имеет долгую историю. В следующие шесть тысяч лет наблюдается эволюция от глиняных таблиц к папирусу, затем к пергаменту и, наконец, к бумаге. Имелось много новшеств в представлении данных: фонетические алфавиты, сочинения, книги, библиотеки, бумажные и печатные издания. Это были большие достижения, но обработка информации в эту эпоху производилась вручную.

0.1. Первое поколение: механизированные менеджеры записей (1900-1955)

Впервые автоматизированная обработка информации появилась приблизительно в 1800 году, когда Джеквард Лум (Jacquard Loom) начал производить раскрой ткани по образцам, представленным перфокартами. Позже аналогичная технология использовалась в

механических пианино.

В 1890 г. Холлерит (Herman Hollerith) использовал технологию перфокарт для выполнения переписи населения Соединенных Штатов. Его система содержала запись для каждой семьи. Каждая запись данных представлялась в виде двоичных структур на перфокарте. Машины сводили подсчеты в таблицы по жилым кварталам, территориальным и административным округам и штатам. Холлерит основал компанию для производства оборудования, для записи данных на карты, сортировки и составления таблиц. Бизнес Холлерита в конце концов привел к возникновению International Business Machines. Эта небольшая компания IBM процветала в период от 1915 до 1960 года как поставщик оборудования регистрации данных для бизнеса и правительственных организаций.

К 1955 году у многих компаний имелись целые этажи, предназначенные для хранения перфокарт, во многом подобно тому, как в шумерских архивах хранились глиняные таблицы. На других этажах размещались шеренги перфораторов, сортировщиков и табуляторов. Эти машины программировались путем перемонтирования управляющих панелей, которые управляли некоторыми регистрами-накопителями и выборочно воспроизводили карты на других картах или на бумаге. Большие компании обрабатывали и производили миллионы записей каждую ночь. Это было бы невозможно при использовании ручных методов обработки. Тем не менее было ясно, что наступает время, когда новая технология вытеснит перфокарты и электромеханические компьютеры.

0.2. Второе поколение: программируемая обработка записей (1955-1970)

Электронные компьютеры с хранимыми программами были разработаны в 1940-х и начале 1950-х годов для выполнения научных и численных вычислений. Примерно в то же время

компания Univac разработала аппаратуру магнитных лент, каждая из которых могла хранить столько информации, сколько десять тысяч перфокарт. Поставка в 1951 году UNIVAC1 в Бюро переписей отразилась на разработке устройств с перфокартами. Эти новые компьютеры могли обрабатывать сотни записей в секунду, и их можно было разместить на гораздо меньшей площади, чем предыдущее оборудование.

Ключевым компонентом этой новой технологии было программное обеспечение. Было сравнительно легко программировать и использовать компьютеры. Было гораздо проще сортировать, анализировать и обрабатывать данные с применением таких языков, как COBOL и RPG. Начали появляться стандартные пакеты для таких общеупотребительных бизнес-приложений как общая бухгалтерия, расчет заработной платы, ведение инвентарных ведомостей, управление подпиской, банковская деятельность и ведение библиотек документов.

Реакция на появление этих новых технологий была предсказуемой. В крупном бизнесе сохраняли еще больше информации и требовали все более и более быстрого оборудования. По мере снижения цен даже в бизнесе средних размеров стали сохранять транзакции на картах и использовать компьютер для обработки карт, поддерживая основной файл на магнитной ленте.

Программное обеспечение этого времени поддерживало модель **обработки записей на основе файлов**. Типичные программы последовательно читали несколько входных файлов и производили на выходе новые файлы. Для облегчения определения этих ориентированных на записи последовательных задач были созданы COBOL и несколько других языков программирования. Операционные системы обеспечивали абстракцию файла для хранения этих записей, язык управления заданиями для выполнения заданий и планировщик заданий

для управления потоком работ.

Системы **пакетной обработки транзакций** сохраняли транзакции на картах или лентах и собирали их в пакеты для последующей обработки. Раз в день эти пакеты транзакций сортировались. Отсортированные транзакции сливались с хранимой на ленте намного большей по размерам базой данных (основным файлом) для производства нового основного файла. На основе этого основного файла также производился отчет, который использовался как гроссбух на следующий бизнес-день. Пакетная обработка позволяла очень эффективно использовать компьютеры, но обладала двумя серьезными ограничениями. Если в транзакции имелась ошибка, она не распознавалась до вечерней обработки основного файла, и могло потребоваться несколько дней для исправления транзакции. Более важным является то, что бизнес не знал текущего состояния базы данных – поскольку транзакции реально не обрабатывались до следующего утра. Для решения этих двух проблем требовался следующий шаг эволюции – оперативные системы. Этот шаг также существенно облегчил написание приложений.

0.3. Третье поколение: оперативные сетевые базы данных (1965-1980)

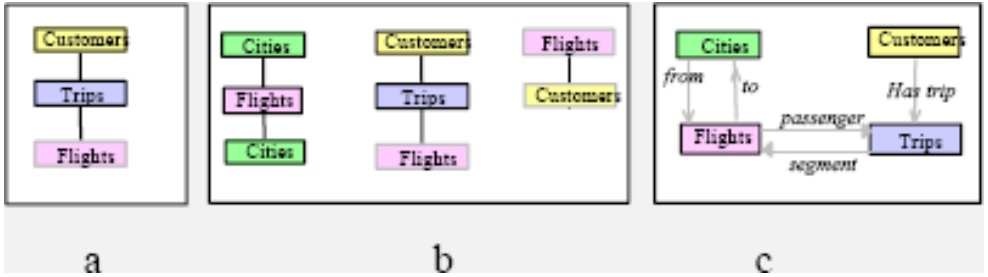
Для таких приложений, как ведение операций на фондовой бирже или резервирование билетов, требуется знание текущей информации. Эти приложения не могут использовать вчерашнюю информацию, обеспечиваемую системами пакетной обработки транзакций, – им нужен немедленный доступ к текущим данным. С конца 1950-х лидирующие компании из нескольких областей индустрии начали вводить в использование системы баз данных с оперативными транзакциями; транзакции над оперативными базами данных обрабатывались в интерактивном режиме. Оперативный доступ к данным обеспечивался несколькими ключевыми технологиями. Аппаратура для подключения к компьютеру интерактивных

компьютерных терминалов прошла путь развития от телетайпов к простым алфавитно-цифровым дисплеям и, наконец, к сегодняшним интеллектуальным терминалам, основанным на технологии персональных компьютеров. **Мониторы телеобработки** представляли собой специализированное программное обеспечение для мультиплексирования тысяч терминалов со скромными серверными компьютерами того времени. Эти мониторы собирали сообщения-запросы, поступающие с терминалов, быстро назначали программы сервера для обработки каждого сообщения и затем направляли ответ на соответствующий терминал. Оперативная обработка транзакций дополняла возможности пакетной обработки транзакций, за которой оставались задачи фонового формирования отчетов.

Оперативные базы данных хранились на магнитных дисках или барабанах, которые обеспечивали доступ к любому элементу данных за доли секунды. Эти устройства и программное обеспечение управления данными давали возможность программам считывать несколько записей, изменять их и затем возвращать новые значения оперативному пользователю. В начале системы обеспечивали простой поиск данных: либо прямой поиск по номеру записи, либо ассоциативный поиск по ключу.

Простые индексно-последовательные организации записей быстро развились к более мощной **модели записей, ориентированной на наборы**. Приложениям часто требуется связать две или более записей. На рис. 2.a показаны некоторые типы записей простой системы резервирования авиационных билетов и их связи. Каждому городу соответствует набор отбывающих рейсов. Каждому заказчику соответствует набор путешествий, а каждое путешествие состоит из набора рейсов. Кроме того, каждому рейсу соответствует набор пассажиров. Как показано на рис. 2.b, эта информация может быть представлена в виде трех иерархий наборов. Каждая из трех иерархий отвечает на отдельный вопрос. Первый – это планирование рейсов в городе. Вторая иерархия дает представление о рейсах заказчика.

Третья иерархия говорит, к какому рейсу относится каждый заказчик. Приложение резервирования билетов нуждается во всех трех этих представлениях данных.



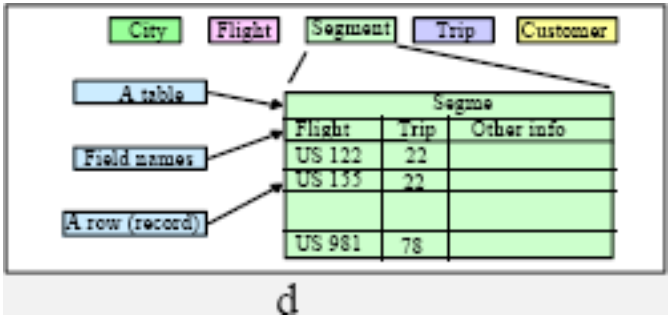


Рис. 2. Эволюция моделей данных. (a) Чистая иерархическая модель с записями, сгруппированными под другими записями. (b) По мере развития приложения разным

пользователям желательно иметь разные представления данных, выражаемые в виде разных иерархий. (c) Диаграмма Бахмана, показывающая типы записей и связи между типами записей. (d) Та же информация, представленная в реляционной модели, где все данные и все связи явно представляются как записи. Отношения показаны вверху рисунка. Некоторые детали отношения Segment показаны в нижнем правом углу; это отношение содержит запись для каждого рейса (сегмента) маршрута пассажира.

Иерархическое представление на рис. 2.b обладает существенным недостатком. Избыточное хранение данных не только дорого стоит, но также и порождает проблемы обновлений: при создании рейса или изменении его информации необходимо обновить данные во всех трех местах (всех трех иерархиях). Для решения этих проблем информацию можно представлять в **сетевой модели данных**, что показано на рис. 2.c. На рис. 2.c изображена одна база данных, в которой каждая запись хранится в одном экземпляре и связывается с набором других записей посредством связи. Например, все рейсы, используемые в путешествии конкретного заказчика, связываются с этим путешествием. Программа может попросить систему баз данных перебрать эти рейсы. При потребности между записями могут создаваться новые связи. Рис. 2.c называется по-разному – диаграммой Бахмана или диаграммой «сущность-связь» [2], [5]. Реляционная диаграмма (рис. 2.d) описывается в следующем разделе.

Управление ассоциативным доступом и обработкой, ориентированной на наборы, было настолько распространено, что в сообществе COBOL выделилась группа Data Base Task Group (DBTG) для определения стандартного способа спецификации таких данных и доступа к ним. Чарльз Бахман (Charles Bachman) в GE (General Electric) построил прототип системы навигации по данным. За руководство работы DBTG, разработавшей стандартный язык определения данных и манипулирования данными, Бахман получил Тьюринговскую премию. В своей Тьюринговской лекции он описал эволюцию моделей плоских файлов к новому миру, где

программы могут осуществлять навигацию между записями, следуя связям между записями [2]. Модель Бахмана напоминает систему Memex Ванневара Буша (Vannevar Bush)[2] или навигационную модель страниц и ссылок сегодняшнего Internet.

В сообществе баз данных COBOL кристаллизовалась концепция **схем** и независимости данных. Они поняли потребность в сокрытии физических деталей расположения записей. Программы должны были видеть только логическую организацию записей и связей, так что программы продолжали работать при изменении и развитии способов хранения данных. Записи, поля и связи, не используемые программой, следовало сокрыть – как по соображениям безопасности, так и для того, чтобы изолировать программу от неизбежных изменений схемы базы данных. В этих ранних базах данных поддерживались три вида схем данных: 1) **логическая схема**, которая определяет глобальный логический проект записей базы данных и связей между записями; 2) **физическая схема**, описывающая физическое размещение записей базы данных на устройствах памяти и в файлах, а также индексы, нужные для поддержки логических связей; 3) предоставляемая каждому приложению **подсхема**, раскрывающая только часть логической схемы, которую использует программа. Механизм логических, физических и подсхем обеспечивал **независимость данных**. И на самом деле, многие программы, написанные в ту эпоху, все еще работают сегодня с использованием той же самой подсхемы, с которой все начиналось, хотя логическая и физическая схемы абсолютно изменились.

Эти оперативные системы должны были решить проблему одновременного выполнения многих транзакций над базой данных, совместно используемой многими терминальными пользователями. До этого подход, основанный на однопрограммном режиме и периодическом изменении основного файла, устранял проблемы параллельного доступа и восстановления. Ранние оперативные системы проложили путь понятию **транзакций**, блокировавших только те

записи, к которым производился доступ. Блокировки позволяют конкурирующим транзакциям получать доступ к разным записям. Системы также поддерживали журнал записей, изменявшихся каждой транзакцией. При сбое транзакции журнал использовался для устранения в базе данных воздействий этой транзакции. Журнал транзакций использовался также для восстановления базы данных в случае аварии носителя. При крахе системы для воссоздания текущего состояния базы данных журнал применялся к ее архивной копии.

К 1980 году сетевые (и иерархические) модели данных, ориентированные на наборы записей, стали очень популярны. Основанная Бахманом компания Cullinet была крупнейшей и наиболее быстро растущей софтверной компанией в мире.

0.4. Четвертое поколение: реляционные базы данных и архитектура клиент-сервер (1980-1995)

Несмотря на успех сетевой модели данных, многие разработчики программного обеспечения чувствовали, что навигационный программный интерфейс был интерфейсом слишком низкого уровня. Было трудно проектировать и программировать эти базы данных. В статье Э.Ф. (Теда) Кодда (E.F. Codd) 1970 года была обрисована реляционная модель [4], которая казалась альтернативой низкоуровневому навигационному интерфейсу. Идея **реляционной модели** состоит в том, чтобы единообразно представлять и сущности, и связи. Реляционная модель данных обладала унифицированным языком для определения данных, навигации по данным и манипулирования данными, а не отдельными языками для каждой из этих задач. Еще более важно то, что реляционная алгебра имеет дело со множествами записей (отношениями) как единым целым, применяя операции к множествам записей целиком и производя множества записей в результате. Реляционные модель данных и операции дают возможность получения более коротких и более простых программ для решения задач управления записями. В

качестве конкретного примера на рис. 2.d показана база данных авиалиний из предыдущего раздела, представленная в виде пяти таблиц. Вместо неявного хранения связи между рейсами и путешествиями в реляционной системе явно хранится каждая пара рейс-путешествие как запись базы данных. Это таблица Segment на рис. 2.d.

Чтобы найти все сегменты, зарезервированные для заказчика Джонса, направляющегося в Сан-Франциско, можно написать следующий запрос на языке SQL:

```
Select      Flight#

From  City, Flight, Segment, Trip, Customer

Where Flight.to = «SF» AND

      Flight.flight# = Segment.flight#  AND

      Segment.trip# = trip.trip# AND

      trip.customer# = customer.customer# AND

      customer.name = «Jones»
```

Эквивалентом этого SQL-запроса на естественном языке является «Найти номера рейсов до Сан-Франциско, которые являются сегментами путешествия любого заказчика с именем Джонс. Для поиска этих рейсов использовать таблицы City, Flight, Segment, Trip и Customer». Эта программа может показаться сложной, но она значительно проще соответствующей навигационной программы.

Получая такой непроцедурный запрос, реляционная система баз данных автоматически

находит лучший способ подбора записей в таблицах City, Flight, Segment, Trip и Customer. Запрос не зависит от того, какие определены связи. Он будет продолжать работать даже после логической реорганизации базы данных. Следовательно, запрос обладает гораздо лучшей независимостью от данных, чем навигационный запрос, основанный на сетевой модели данных. Вдобавок к улучшенной независимости данных реляционные программы часто в пять – десять раз проще соответствующих навигационных программ.

Вдохновляемые идеями Кодда в течение 1970-х исследователи из академии и индустрии экспериментировали с этим новым подходом к структуризации баз данных и обеспечения доступа к ним, обещавшим более простое моделирование данных и прикладное программирование. Многие реляционные прототипы, разработанные в течение этого периода, сошлись на общей модели и языке. Исследования в IBM Research, возглавлявшиеся Тедом Коддом, Реймондом Бойсом (Raymond Boyce) и Доном Чемберлином (Don Chamberlin), и работа в Калифорнийском университете г. Беркли, которой руководил Майкл Стоунбрейкер (Michael Stonebraker), породили язык, названный SQL. Этот язык был впервые стандартизован в 1985 году. Впоследствии были произведены два основных расширения стандарта [5], [6]. Сегодня почти все системы баз данных обеспечивают интерфейс SQL. Кроме того, во всех системах поддерживаются собственные расширения, выходящие за рамки этого стандарта.

Реляционная модель обладает некоторыми неожиданными преимуществами, кроме повышения продуктивности программистов и простоты использования. Реляционная модель оказалась хорошо пригодной к использованию в архитектуре клиент-сервер, к параллельной обработке и графическим пользовательским интерфейсам.

Клиент-серверная архитектура. Приложение клиент-сервер разбивается на две части. **Клиентская** часть отвечает за поддержку ввода и представление выводных данных для

пользователя или клиентского устройства. **Сервер** отвечает за хранение базы данных, обработку клиентских запросов к базе данных, возврат клиенту общего ответа. Реляционный интерфейс особенно удобен для использования в архитектуре клиент-сервер, поскольку приводит к обмену высокоуровневыми запросами и ответами. Высокоуровневый интерфейс SQL минимизирует коммуникации между клиентом и сервером. Сегодня многие клиент-серверные средства строятся на основе протокола Open Database Connectivity (ODBC), который обеспечивает для клиента стандартный механизм запросов высокого уровня к серверу. Парадигма клиент-сервер продолжает развиваться. Как разъясняется в следующем разделе, имеется возрастающая тенденция интеграции процедур в серверах баз данных. В частности, такие процедурные языки, как BASIC и Java, были добавлены к серверам, чтобы клиенты могли вызывать прикладные процедуры, выполняемые на сервере.

Параллельная обработка баз данных была вторым неожиданным преимуществом реляционной модели. Отношения являются однородными множествами записей. Реляционная модель включает набор операций, замкнутых по композиции: каждая операция получает отношения на входе и производит отношение как результат. Поэтому реляционные операции естественным образом предоставляют возможности конвейерного параллелизма путем направления вывода одной операции на вход следующей. Редко удается найти длинные конвейеры, но часто реляционные операции могут быть разделены таким образом, что образуется *N* клонов каждой операции и каждый клон может работать с одной *N*-й частью отношения-операнда. Пути применения этих идей были проложены в академическом сообществе и компанией Teradata Corporation (теперь NCR). Сегодня распространенной практикой реляционных систем является стократное увеличение скорости за счет параллелизма. Задачи интеллектуального анализа данных (data mining), для решения которых могли бы потребоваться недели или месяцы поиска в многотерабайтных базах данных, при

использовании параллелизма выполняются за часы. Этот параллелизм полностью автоматизирован. Проектировщики только поставляют данные системе баз данных, и сама система разделяет и индексирует данные. Пользователи предоставляют системе запросы (на основе ODBC), и система автоматически обеспечивает параллельный план выполнения запроса и выполняет его.

Реляционные данные также хорошо приспособлены к **графическим пользовательским интерфейсам** (GUI). Очень легко представлять отношение как множество записей – к отношениям пригодна метафора электронных таблиц. Пользователи могут просто создавать отношения в виде электронных таблиц и визуально манипулировать ими. На самом деле имеется много инструментальных средств, позволяющих перемещать данные между документами, электронными таблицами и базами данных. Явно и единообразно представленные данные, связи и метаданные делают это возможным.

Реляционные системы, объединенные с GUI, позволяют сотням тысяч людей каждый день формулировать сложные запросы к базам данных. Комбинация GUI и реляционных систем подходит ближе всего в достижению цели автоматического программирования. GUI позволяют просто конструировать сложные запросы. Для заданного непроцедурного запроса реляционные системы находят наиболее эффективные способы его выполнения.

Если продолжать отслеживать историю, можно отметить, что к 1980 году Oracle, Informix и Ingress принесли на рынок свои системы управления реляционными базами данных. Через несколько лет на рынке появились продукты IBM и Sybase. К 1990 году реляционные системы стали более популярными, чем предшествовавшие им ориентированные на наборы записей навигационные системы. Между тем файловые системы и системы, ориентированные на наборы, оставались рабочими лошадками многих корпораций. С годами эти корпорации

построили громадные приложения и не могли легко перейти к использованию реляционных систем. Реляционные системы скорее стали ключевым средством для новых клиент-серверных приложений.

0.5. Пятое поколение: мультимедийные базы данных (1995-...)

Реляционные системы внесли много усовершенствований в удобство использования, графические интерфейсы, клиент-серверные приложения, распределенные базы данных, параллельный поиск данных и интеллектуальный анализ данных. Тем не менее, около 1985 года исследовательское сообщество начало рассматривать вопросы, выходящие за рамки реляционной модели. Традиционно существовало четкое разделение программ и данных. Этот подход хорошо работал, пока речь шла только о таких данных, как числа, символы, массивы, списки или множества записей. По мере появления новых приложений разделение программ и данных стало проблематичным. Приложениям требовалось дать данным поведение. Например, если данные представляли сложный объект, то методы поиска, сравнения и манипулирования данными становились специфичными для типов данных документов, изображений, звука или карта (см. рис. 3).

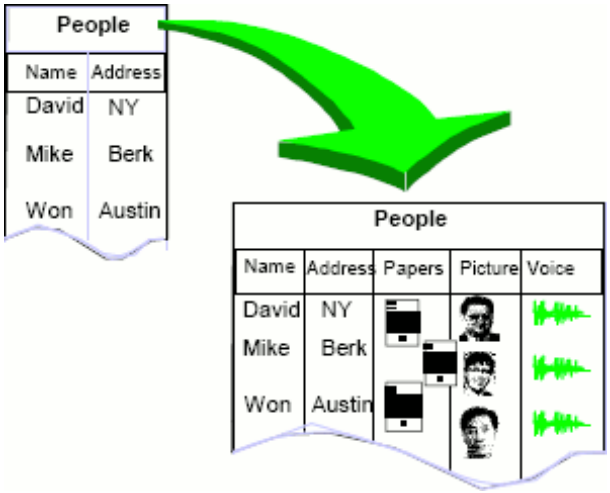


Рис. 3. Переход от традиционных баз данных, хранящих числа и символы, к объектно-реляционным базам данных, где каждая запись может содержать данные со сложным поведением. Это поведение может быть инкапсулировано в библиотеках классов, которые поддерживают новые типы. В этой модели система баз данных хранит и выбирает данные и обеспечивает связи между элементами данных, а библиотеки классов обеспечивают поведение этих элементов.

Традиционный подход заключался во встраивании типов данных прямо в систему баз данных. В языке SQL были добавлены новые типы данных для времени, временных интервалов и символьных строк с двухбайтовым представлением символов. Каждое из этих расширений

было значительным достижением. Когда они были сделаны, результаты удовлетворили не всех. Например, в SQL тип данных «время» не позволяет представлять даты до Рождества Христова, а в символьных строках нельзя применять кодировку Unicode (универсальный набор символов почти для всех языков). Пользователи, желающие использовать Unicode или даты до нашей эры, должны определять свои собственные типы данных. Эти простые примеры, а также многие другие убедили сообщество баз данных в том, что системы баз данных должны разрешать прикладным специалистам реализовывать типы данных для своих прикладных областей. Географам следует иметь возможность реализации карт, специалистам в области текстов имеет смысл реализовывать индексацию и выборку текстов, специалистам по изображениям стоило бы реализовать библиотеки типов для работы с изображениями. Конкретным примером может служить распространенный объектный тип временных рядов. Вместо встраивания этого объекта в систему баз данных рекомендуется реализация соответствующего типа в виде библиотеки классов с методами для создания, обновления и удаления временных рядов. Дополнительные методы суммируют отклонения и интерполируют точки в рядах, сравнивают, комбинируют и вычитают временные ряды. После построения такой библиотеки классов она может быть «вставлена» в любую систему баз данных. Система баз данных будет хранить объекты этого типа и управлять данными (безопасность, параллельный доступ, восстановление и индексирование), но содержимым и поведением объектов временных рядов будет заведовать тип данных.

Люди из сообщества объектно-ориентированного программирования четко распознали проблему: для разработки типов данных требуется хорошая модель данных и унификация процедур и данных. Действительно, программы инкапсулируют данные и обеспечивают все методы для манипулирования данными. Исследователи, начинающие и установившиеся поставщики реляционных баз данных долго и упорно работали, начиная с 1985 года, чтобы

либо заменить реляционную модель, либо объединить объектно-ориентированные и реляционные системы. В конце 1980-х на рынке появилось более десяти продуктов объектно-ориентированных баз данных, но заказчики не спешили принять эти системы. Тем временем традиционные поставщики старались расширить язык SQL, чтобы включить в него концепции объектно-ориентированного подхода, сохраняя преимущества реляционной модели.

Все еще ведутся горячие дебаты относительно сравнительных преимуществ эволюции и революции в моделях данных. Нет спора, что системы баз данных должны хранить и извлекать объекты, управляемые библиотеками классов. Споры относятся к роли SQL, деталям объектной модели и базовым библиотекам классов, которые следует поддерживать в системах баз данных.

Быстрое развитие Internet усиливает эти дебаты. Клиенты и серверы Internet строятся с использованием апплетов и «хелперов», которые сохраняют, обрабатывают и отображают данные того или иного типа. Пользователи вставляют эти апплеты в браузер или сервер. Общераспространенные апплеты управляют звуком, изображениями, текстами, видео, электронными таблицами, графами. Для каждого из ассоциированных с этими апплетами типов данных имеется библиотека классов. Настольные компьютеры и Web-браузеры являются распространенными источниками и приемниками большей части данных. Поэтому типы и объектные модели, используемые в настольных компьютерах, будут диктовать, какие библиотеки классов должны поддерживаться на серверах баз данных.

Подводя итог, заметим, что базы данных призваны хранить больше, чем только числа и текстовые строки. Они используются для хранения многих видов объектов, которые мы видим в World Wide Web, и связей между этими объектами. Различие между базой данных и остальной частью Web становится неясным. Каждый поставщик баз данных обещает

«универсальный сервер», который будет хранить и анализировать все формы данных (все библиотеки классов и соответствующие объекты).