Лекция 1. Общее представление об информационной системе

В уже достаточно долгой истории компьютерной индустрии (скоро будем отмечать 50-летний юбилей) всегда можно было выделить два основных направления: вычисления, а также накопление и обработка информации. Как известно, возникновение компьютеров главным образом стимулировалось необходимостью проведения массивных расчетов для создания ядерного оружия и ракетной техники. Объемы требуемых вычислений просто не позволяли произвести их в приемлемое время традиционным коллективом расчетчиков. Итак, первыми пользователями компьютеров и разработчиками компьютерных программ стали вычислительные математики. До сих пор многие представители старшего поколения программистов предпочитают называть себя математиками, даже если в последние 20-30 лет им не приходилось написать хотя бы одну вычислительную программу, не говоря уже о разработке методов и алгоритмов компьютерных вычислений.

Однако, почти сразу на появление компьютеров обратили внимание бизнесмены. Как правило, в гражданском бизнесе не требуются массивные расчеты за исключением таких отраслей как, например, авиа- или автомобилестроение. В более распространенных видах гражданского бизнеса (банковское дело, биржевые операции, системы резервирования билетов или мест в гостиницах) основной проблемой всегда являлись объемы информации, которую необходимо собирать, надежно хранить и оперативно обрабатывать. Появление информационных систем, основным назначением которых является решение отмеченной проблемы, явилось ответом компьютерной индустрии на требования мира бизнеса.

В этой лекции будет введено общее представление о сути понятия *"информационная система"*. Основной целью является формулировка понятийной основы.

Специфика информационных программных систем

Конечно, в зависимости от конкретной области применения информационные системы могут очень сильно различаться по своим функциям, архитектуре, реализации. Однако можно выделить, по крайней мере, два свойства, которые являются общими для всех информационных систем. Во-первых, любая информационная система предназначена для сбора, хранения и обработки информации. Поэтому в основе любой информационной системы лежит среда хранения и доступа к данным. Среда должна обеспечивать уровень надежности хранения и эффективность доступа, которые соответствуют области применения информационной системы. Заметим, что в вычислительных программных системах наличие такой среды не является обязательным. Основным требованием к программе, выполняющей численные расчеты (если, конечно, говорить о решении действительно серьезных задач), является ее быстродействие. Нужно, чтобы программа произвела достаточно точные результаты за установленное время. При решении серьезных вычислительных задач даже на суперкомпьютерах это время может измеряться неделями, а иногда и месяцами. Поэтому программисты-вычислители всегда очень скептически относятся к хранению данных во внешней памяти, предпочитая так организовывать программу, чтобы в течение как можно более долгого времени обрабатываемые данные помещались в основной памяти компьютера. Внешняя память обычно используется для периодического (нечастого) сохранения промежуточных результатов вычислений, чтобы в случае сбоя компьютера можно было продолжить работу программы от сохраненной контрольной точки.

Во-вторых, информационные системы ориентируются на конечного пользователя, например, банковского клерка. Такие пользователи могут быть очень далеки от мира

компьютеров. Для них терминал, персональный компьютер или рабочая станция представляют собой всего лишь орудие их собственной профессиональной деятельности. Поэтому информационная система обязана обладать простым, удобным, легко осваиваемым интерфейсом, который должен предоставить конечному пользователю все необходимые для его работы функции, но в то же время не дать ему возможность выполнять какие-либо лишние действия. Иногда этот интерфейс может быть графическим с меню, кнопками, подсказками и т.д. Сейчас очень популярны графические интерфейсы, многие современные средства разработки информационных приложений прежде всего ориентированы на разработку графических интерфейсов. С другой стороны, немного странным фактом является то, что многие конечные пользователи (например, банковские операционисты) не любят графические терминалы, предпочитая более убогие интерфейсные средства доступа к информационной системе с современного, но традиционного алфавитно-цифрового терминала. Это кажется действительно несколько странным, потому что на Западе, где практически любой кассовый аппарат является в действительности персональным компьютером, невозможно увидеть ни одного алфавитно-цифрового монитора. Возможно, в России это просто временный социально-психологический эффект: после многих лет общения с низкокачественными отечественными цветными телевизорами люди считают, что использовать графический монитор более вредно, чем монохромный алфавитно-цифровой. Но в любом случае наличие развитых интерфейсных средств является обязательным для любой современной информационной системы.

И снова заметим, что вычислительные программные системы не обязательно обладают развитыми интерфейсами. Конечно, это зависит от степени отчуждаемости программного продукта. Если система предназначена для продажи, то она должна обладать хорошим интерфейсом хотя бы в целях маркетинга. Но как правило, серьезные вычислительные программы почти уникальны. Расчеты выполняются либо разработчиками программ, либо людьми из того же окружения. Для них гораздо важнее быстродействие вычислений, чем удобство запуска программы, а наличие развитого интерфейса предполагает существенный расход компьютерных ресурсов. Как профессионалы компьютерного мира, эти люди могут справиться с некоторыми неудобствами при работе с компьютером.

Задачи информационных систем

Конкретные задачи, которые должны решаться информационной системой, зависят от той прикладной области, для которой предназначена система. Области применения информационных приложений разнообразны: банковское дело, страхование, медицина, транспорт, образование и т.д. Трудно найти область деловой активности, в которой сегодня можно было обойтись без использования информационных систем. С другой стороны, очевидно, что, например, конкретные задачи, решаемые банковскими информационными системами, отличаются от задач, решение которых требуется от медицинских информационных систем.

Но можно выделить некоторое количество задач, не зависящих от специфики прикладной области. Естественно, такие задачи связаны с общими чертами информационных систем. Прежде всего, кажется бесспорным мнение о том, что наиболее существенной составляющей является информация, которая долго накапливается и утрата которой невосполнима.

В качестве примера рассмотрим ситуацию, существующую в Зеленчукской астрофизической лаборатории. В этой лаборатории в горах в районе Нижнего Архыза установлен один из крупнейших в мире зеркальных телескопов (диаметр зеркала - 6

метров). Уникальные природные условия этого района Северного Кавказа позволяют максимально эффективно использовать возможности обсерватории. В самом Зеленчуке имеется крупнейший в России радиотелескоп. Комбинированное использование этих ресурсов в течение многих лет (более 10) позволило астрофизикам накопить уникальную информацию относительно разного рода космических объектов. К сожалению, компьютерные возможности лаборатории в первые годы ее существования были весьма ограничены, и поэтому накапливаемые данные хранились в основном на магнитных лентах. Известно, что любой магнитный носитель стареет, а магнитные ленты еще и пересыхают. В результате основной проблемой группы поддержки информационных ресурсов уже несколько лет является копирование старых магнитных лент на новые носители. Старые ленты часто не читаются, и приходится тратить громадные усилия и средства для их реанимирования. Здесь уже не до создания информационной системы. Успеть бы спасти информацию. Хотя, конечно, астрофизикам очень нужны информационные системы, позволяющие хотя бы частично автоматизировать огромные объемы работ по анализу и обобщению накопленной информации. Основной вывод, который можно сделать на основе этой нравоучительной истории, состоит в том, что если некоторая организация планирует долговременное накопление ценной информации, то с самого начала должны быть обдуманы надежные способы ее долговременного хранения. В частности, информация, накопленная Зеленчукской лабораторий, должна храниться вечно.

Конечно, уровень надежности и продолжительность хранения информации во многом определяются конкретными требованиями корпорации к информационной системе. Например, можно представить себе малую торговую компанию с быстрым оборотом, в информационной складской системе которой достаточно поддерживать информацию о товарах, имеющихся на складе, и об еще неудовлетворенных заявках от потребителей. Но кто знает, не потребуется ли впоследствии полная история работы склада с момента основания компании.

Следующей задачей, которую должно выполнять большинство информационных систем, - это хранение данных, обладающих разными структурами. Трудно представить себе более или менее развитую информационную систему, которая работает с одним однородным файлом данных. Более того, разумным требованием к информационной системе является то, чтобы она могла развиваться. Могут появиться новые функции, для выполнения которых требуются дополнительные данные с новой структурой. При этом вся накопленная ранее информация должна остаться сохранной. Теоретически можно решить эту задачу путем использования нескольких файлов внешней памяти, каждый из которых хранит данные с фиксированной структурой. В зависимости от способа организации используемой системы управления файлами эта структура может быть структурой записи файла (как, например, в файловых системах DEC VMS) или поддерживаться отдельной библиотечной функцией, написанной специально для данной информационной системы (если, конечно, не удастся найти подходящую функцию в одной из существующих библиотек). Ко второму способу решения проблемы пришлось бы прибегать при работе в среде ОС UNIX.

При использовании такого подхода информационная система перегружается деталями организации хранилища данных. При выполнении функций уровня пользовательского интерфейса информационной системе самой приходится выполнять выборку информации из файлов по заданному критерию. Некоторые системы управления файлами позволяют выбирать записи по простому критерию, например, по заданному значению ключа записи. Но, во-первых, такие возможности выборки всегда ограничены, и с большой вероятностью придется вынести хотя бы часть функций выборки в код самой

информационной системы. Во-вторых, наличие нескольких файлов данных разной структуры неявно предполагает, что при выполнении некоторых функций информационной системы потребуется выборка согласованной (по заданному критерию) информации из нескольких файлов. Такие возможности никогда не поддерживаются файловыми системами.

Известны примеры реально функционирующих информационных систем, в которых хранилище данных планировалось основывать на файлах. В результате развития большинства таких систем в них выделился отдельный компонент, который представляет собой примитивную разновидность системы управления базами данных (СУБД). Самодельные СУБД - главный бич информационных систем. Поначалу кажется, что все очень просто: набор возможных запросов становится известным при проектировании информационной системы; для каждого типа запроса можно придумать эффективный способ выполнения запроса. После этого остается простая программистская работа, и специализированная СУБД готова. Однако, потом оказывается, что не все возможные запросы были учтены при проектировании. Бедный разработчик СУБД постоянно добавляет в нее новые функции, пока не решает создать общий язык запросов, на котором можно сформулировать любой запрос к базе данных соответствующей информационной системы. Через некоторое время в корпорации принимают решение разработать еще одну информационную систему, структуры хранимых данных которой, естественно, отличаются от тех, что были в базе данных первой информационной системы. Что же, делать еще одну специализированную СУБД? Нет, говорит начальство. У нас уже есть одна. Давайте попробуем применить ее. И это приводит к тому, что наивный самодельщик вынужден сделать простую (скорее всего, персональную) СУБД общего назначения, которая может получить из базы данных информацию о структуре ее файлов (т.е. в базе данных хранятся теперь еще и метаданные, определяющие структуры обычных данных, - схема базы данных), а также выполнить произвольный запрос к этой базе данных. В результате, даже если удается добиться работоспособности разработанной СУБД, это означает всего лишь изобретение еще одного велосипеда, поскольку СУБД такого уровня существует великое множество. Они дешевы и поддерживаются производителями.

До сих пор мы говорили о тех функциях информационной системы, которые требуют выборки данных из внешнего хранилища, например, производят отчеты. Но откуда берутся данные во внешнем хранилище? Каким образом поддерживается соответствие хранимой информации состоянию предметной области? Конечно, для этого должны существовать дополнительные функции информационной системы, которые обеспечивают ввод, обновление и удаление данных. Поддержка этих функций существенно повышает уровень требований к СУБД.

Если говорить о групповых или корпоративных информационных системах, то их наличие предполагает возможность работы с системой с нескольких рабочих мест. Некоторые из конечных пользователей изменяют содержимое базы данных (вводят, обновляют, удаляют данные). Другие выполняют операции, связанные с выборкой из базы данных. Третьи делают и то, и другое. Вся проблема состоит в том, что такая коллективная работа должна производиться согласованно и желательно, чтобы согласованность действий обеспечивалась автоматически. Под согласованностью действий мы понимаем то, что оператор, формирующий отчеты, не сможет воспользоваться данными, которые начал, но еще не закончил формировать другой оператор. Оператор формирующий данные, не сможет выполнить операцию над данными, которыми пользуется другой оператор, начавший, но не закончивший формировать отчет. Оператор, желающий обновить или удалить данные, не сможет выполнить операцию до тех пор, пока не закончится аналогичная операция над теми же данными, которую ранее начал, но еще не закончил

другой оператор. При поддержке согласованности действий все результаты, получаемые от информационной системы, будут соответствовать согласованному состоянию базы данных, т.е. будут достоверны и непротиворечивы.

Подобные рассуждения вызвали появления понятия *классической транзакции*. Будем понимать под целостным состоянием базы данных информационной системы такое ее состояние, которое соответствует требованиям прикладной области (или, вернее, требованиям модели прикладной области, на основе которой проектировалась информационная система). Тогда классической транзакцией называется последовательность операций изменения базы данных и/или выборки из базы данных, воспринимаемая СУБД как атомарное действие. Это означает, что при успешном завершении транзакции СУБД гарантирует наличие в базе данных результатов всех операций изменения, произведенных при выполнении транзакции. Условием успешного завершения транзакции является то, что база данных находится в целостном состоянии. Если это условие не выполняется, то СУБД производит полный откат транзакции, ликвидируя в базе данных результаты всех операций изменения, произведенных при выполнении транзакции. Тем самым, легко видеть, что база данных будет находиться в целостном состоянии при начале любой транзакции и останется в целостном состоянии после успешного завершения любой транзакции.

Все развитые СУБД поддерживают понятие транзакции. Если информационная система базируется на СУБД такого класса, то для обеспечения согласованности действий параллельно работающих конечных пользователей достаточно при проектировании системы правильно связать операции информационной системы с транзакциями СУБД. Это относится уже к области проектирования, и мы подробно обсудим проблемы в следующих частях курса.

Еще одно небольшое замечание относительно транзакций. СУБД может очень просто обрабатывать транзакции, выполняя их последовательно. Этого достаточно, чтобы обеспечить согласованность действий "параллельно" работающих операторов. Но реальной параллельности в этом случае, конечно, не будет. СУБД выстроит всех пользователей в общую очередь и будет пропускать по-одному даже если они вовсе не конфликтуют по данным. Развитые СУБД так не работают. Они стремятся максимально перемешивать запросы и операторы изменения базы данных, поступающие от разных транзакций, с тем лишь условием, что конечный результат выполнения всего набора транзакций будет эквивалентен результату их некоторого последовательного выполнения. В мире баз данных такая политика СУБД называется *политикой полной сериализации смеси транзакций*. Очевидно, что полная сериализация транзакций достаточна для достижения согласованности действий теперь уже действительно параллельной работы операторов информационной системы. Но полная сериализация транзакций не всегда является необходимой для требуемой согласованности действий. Существуют модели ослабленной сериализации, которая допускает еще большую параллельность и вызывает меньшие накладные расходы.

На первый взгляд кажется, что понятие транзакции чуждо персональным СУБД, с которыми в любой момент времени работает только один пользователь. Однако рассмотрим еще раз упоминавшуюся задачу надежного хранения данных. Что это означает более конкретно? Теперь, после того, как мы ввели понятия целостного состояния базы данных и транзакции, под надежностью хранения данных мы понимаем гарантию того, что последнее по времени целостное состояние базы данных будет сохранено СУБД при любых обстоятельствах. Одно такое возможное обстоятельство мы уже упоминали: нарушение целостности базы данных при окончании транзакции.

Традиционное решение - откат транзакции. Второй возможный случай - аварийное выключение питания, в результате чего теряется содержимое основной памяти, в буферах которой, возможно, находились измененные, но еще не записанные во внешнюю память блоки базы данных. Традиционное решение - откат всех транзакций, которые не завершились к моменту аварии, и гарантированная запись во внешней памяти результатов завершившихся транзакций. Естественно, это можно сделать только после возобновления подачи питания в ходе специальной процедуры восстановления. Наконец, третий случай - авария внешнего носителя базы данных. Традиционное решение - переписать на исправный внешний носитель архивную копию базы данных (конечно, нужно ее иметь), после чего повторить операции всех транзакций, которые были выполнены после архивации, а затем выполнить откат всех транзакций, не закончившихся к моменту аварии. С разными модификациями развитые СУБД обеспечивают решение этих проблем за счет поддержки дополнительного файла внешней памяти - журнала базы данных. В журнал помещаются записи, соответствующие каждой операции изменения базы данных, а также записи о начале и конце каждой транзакции. Файл журнала требует особой надежности хранения (пропадет журнал - базу данных не восстановишь), что обычно достигается путем поддержки зеркальной копии. Вернемся к началу этого абзаца. Разве надежность хранения данных не нужна персональным информационным системам, если, конечно, они не совсем примитивны? Как мы видели, надежности хранения невозможно добиться, если не поддерживать в СУБД понятие транзакции. К сожалению, до последнего времени в большинстве персональных СУБД транзакции не поддерживались (само собой отсутствовали и средства определения и поддержки целостности баз данных). Поэтому о надежности хранения информации в информационных системах, основанных на персональных СУБД, можно говорить только условно.

В корпоративных информационных системах по естественным причинам часто возникает потребность в распределенном хранении общей базы данных. Например, разумно хранить некоторую часть информации как можно ближе к тем рабочим местам, в которых она чаще всего используется. По этой причине при построении информационной системы приходится решать задачу согласованного управления распределенной базой данных (иногда применяя методы репликации данных). При однородном построении распределенной базы данных (на основе однотипных серверов баз данных) эту задачу обычно удается решить на уровне СУБД (большинство производителей развитых СУБД поддерживает средства управления распределенными базами данных). Если же система разнородна (т.е. для управления отдельными частями распределенной базы данных используются разные серверы), то приходится прибегать к использованию вспомогательных инструментальных средств интеграции разнородных баз данных типа мониторов транзакций.

Традиционным методом организации информационных систем является двухзвенная архитектура "клиент-сервер" (рисунок 1). В этом случае вся прикладная часть информационной системы выполняется на рабочих станциях системы (т.е. дублируется), а на стороне сервера(ов) осуществляется только доступ к базе данных. Если логика прикладной части системы достаточно сложна, то такой подход порождает проблему "толстого" клиента. Каждая рабочая станция должна обладать достаточным набором ресурсов, чтобы быть в состоянии произвести прикладную обработку данных, поступающих от пользователя и/или из базы данных. Для того, чтобы клиенты могли быть "тощими", а зачастую и для повышения общей эффективности системы, все чаще применяются трехзвенные архитектуры "клиент-сервер" (рисунок 2). В этой архитектуре, кроме клиентской части системы и сервера(ов) базы данных, вводится промежуточный сервер приложений. На стороне клиента выполняются только интерфейсные действия, а вся логика обработки информации поддерживается в сервере приложений. В следующих

частях курса мы рассмотрим возможные технологии организации трехзвенных архитектур.

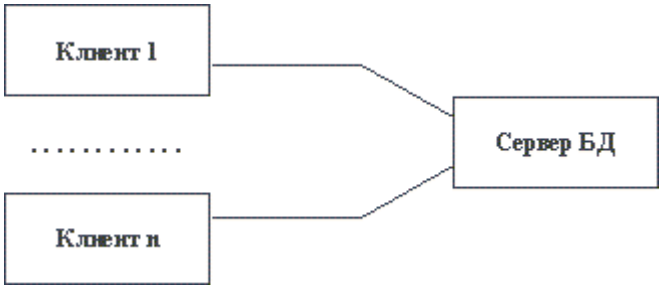

Рис. 1 Традиционная двухзвенная архитектура "клиент-сервер"

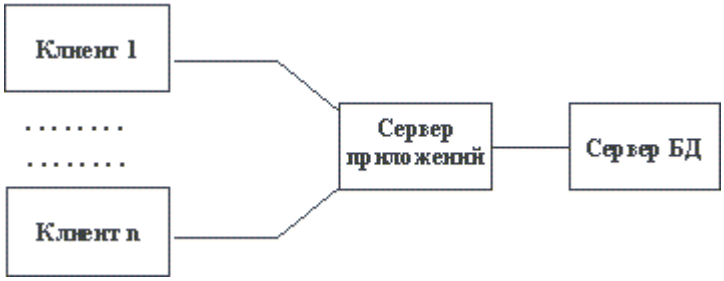

Рис. 2. Трехзвенная архитектура "клиент-сервер" с выделенным сервером приложений

Заметим, что некоторые черты трехзвенности могут присутствовать и в двухзвенной архитектуре. Если, например, используемый сервер баз данных поддерживает развитый механизм хранимых процедур (например, такой, как в Oracle V.7), то можно перебросить некоторую часть логики приложения на сторону баз данных. Заметим, что механизм хранимых процедур недостаточно полно специфицирован в стандарте языка SQL. Как только вы решаетесь использовать действительно развитые средства, то немедленно привязываете свою информационную систему к конкретному производителю серверов баз данных. Развязаться будет очень трудно.

И наконец, еще один класс задач относится к обеспечению удобного и соответствующего целям информационной системы пользовательского интерфейса. Более или менее просто выяснить функциональные компоненты интерфейса, например, какого вида должны предлагаться формы и какого вида должны выдаваться отчеты. Но построение действительно удобного и неутомительного для пользователя интерфейса - это задача дизайнера интерфейса. Простой аналог: при наличии полного набора качественной мебели хороший дизайнер сможет красиво оформить удобную для жизни квартиру (все на месте и под рукой). Плохой же дизайнер, скорее всего, добьется лишь того, что сможет запихнуть в квартиру всю мебель, а потом хозяину квартиры все время будет казаться, что у него слишком много ненужной мебели и ничего невозможно найти. Нужно отдавать себе отчет в том, что задача эргономичности интерфейса не формализуется. При ее решении не помогут никакие средства автоматизации разработки интерфейса. Такие средства облегчают только построение компонентов интерфейса. Построение же полного интерфейса - это творческая задача, при решении которой нужно учитывать требования эстетичности и удобства, а также принимать во внимание особенности конкретной области применения информационной системы.

На первый взгляд упомянутая задача кажется не очень существенной. Можно полагать, что если информационная система обеспечивает полный набор функций и ее интерфейс

обеспечивает доступ к любой из этих функций, то конечные пользователи должны быть удовлетворены. На самом деле, это не так. Пользователи часто судят о качестве системы в целом исходя из качества ее интерфейса. Более того, эффективность использования системы зависит от качества интерфейса. Поскольку, как отмечалось выше, задача построения эргономичного интерфейса не формализуется, мы больше не будем затрагивать ее в этом курсе, а ограничимся рассмотрением вопросов построения компонентов интерфейса.